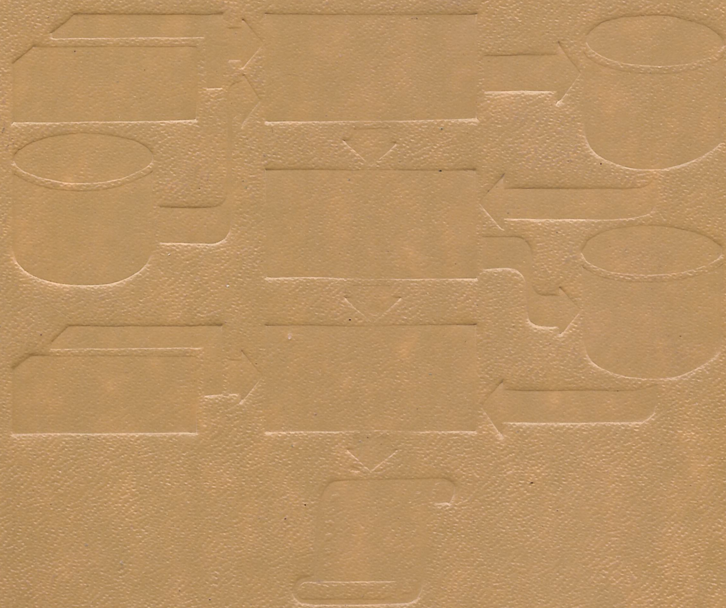
МАКРОПРОГРАММИРОВАНИЕ В ВЫЧИСЛИТЕЛЬНОЙ СРЕДЕ ДОС/ЕС

Л. И. Шатровский

МАКРОПРОГРАММИРОВАНИЕ В ВЫЧИСЛИТЕЛЬНОЙ СРЕДЕ ДОС/ЕС



Л. И. Шатровский

МАКРОПРОГРАММИРОВАНИЕ В ВЫЧИСЛИТЕЛЬНОЙ СРЕДЕ ДОС/ЕС

Под редакцией проф. К. А. ПУПКОВА



МОСКВА «СОВЕТСКОЕ РАДИО» 1979

ББК 32.81
Ш 29
УДК 681.3.06

Шатровский Л. И. **Макропрограммирование в вычислительной среде ДОС/ЕС.**— М.: Сов. радио, 1979—240 с.

В книге излагается Язык Управления Заданиями ДОС/ЕС, операторы системных программ «РЕДАКТОР» и «БИБЛИОТЕКАРЬ», что позволяет хранить в памяти машины как готовые программы, так и находящиеся в процессе разработки их части, строить комплексы программ, в том числе программ оверлейной структуры из сравнительно простых программных модулей, разрабатываемых различными программистами.

В книге описываются также языки высокого уровня, допускающие использование в Дисковой Операционной Системе, наиболее широко распространенной на младших моделях ЕС ЭВМ. Изложение этих языков ведется в сравнении их с языком АЛГОЛ-60 с помощью Нормальной формы Бэкуса. Описывается методика отладки программ с помощью диагностических сообщений трансляторов, генерируемых системой как на этапе трансляции программ, так и на этапе их выполнения.

Книга содержит примеры, помогающие усвоению материала. Она может служить пособием программисту, знакомому с каким-либо алгоритмическим языком или имеющему некоторый опыт программирования на ЭВМ других систем, при овладении навыками работы с ЕС ЭВМ. Книга окажется также полезной разработчикам программного обеспечения ИПС, АСУ и других программных систем, базирующихся на ЭВМ Единой Системы.

Рис. 20, табл. 26, библ. 5 назв.

Редакция кибернетической литературы

Ш $\frac{30502\text{-}012}{046(01)\text{-}79}$ 74-78 1502000000

Введение

Широкое внедрение вычислительной техники в народное хозяйство и другие сферы общественной жизни является ярким показателем заботы Коммунистической партии и Советского правительства об улучшении условий труда советских людей. «Резкое сокращение доли ручного труда, комплексная механизация и автоматизация производства становятся непременным условием экономического роста» — сказал тов. Л. И. Брежнев в отчетном докладе ЦК КПСС XXV съезду партии[1]. Об этом же говорится в Конституции СССР: «Государство заботится об улучшении условий и охране труда, его научной организации, о сокращении, а в дальнейшем и полном вытеснении тяжелого физического труда на основе комплексной механизации и автоматизации производства»[2].

Рост выпуска средств вычислительной техники в IX пятилетке в 4,3 раза по сравнению с VIII и намечаемый на X пятилетку дальнейший рост еще в 1,8 раза с одновременным переходом на выпуск ЭВМ третьего поколения заставляет по-новому подходить к подготовке основной категории специалистов, от которых зависит эффективное использование ЭВМ,— программистов.

Программисту зачастую достаточно было владеть одним из алгоритмических языков ЭВМ второго поколения, например АЛГОЛом-60 или даже языком машинных кодов, чтобы успешно пользоваться ЭВМ второго поколения. Для работы с ЭВМ третьего поколения программист должен иметь несравненно более высокую квалификацию. И объясняется это не недостатками их структуры, а многократным усложнением задач, решаемых на ЭВМ,— задач, не доступных машинам предыдущих поколений.

---

[1] Материалы XXV съезда КПСС. М., «Политиздат», 1975, с. 43.

[2] Конституция (Основной закон) Союза Советских Социалистических Республик. Ст. 21.1977.

Современные ЭВМ — это уже не прежние машины, содержащие один процессор и небольшое, почти стабильное количество периферийных устройств. Теперь периферийные устройства в свою очередь являются небольшими ЭВМ, содержащими собственные процессоры, устройства управления и др. ЭВМ превратились в *вычислительные системы*, состоящие из одного или нескольких *центральных* процессоров и большого количества периферийных устройств, используемых как отдельными процессорами, так и системой в целом. Вычислительные системы могут комплектоваться удаленными терминалами — *абонентскими пунктами*, через которые пользователь может иметь доступ к ресурсам системы.

Вся эта аппаратура должна согласованно работать, а ее количество и быстродействие не позволяют надеяться, что это согласование обеспечит оператор системы, находящийся за главным пультом. В целях увеличения производительности ЭВМ часть функций оператора передается специальной программе — супервизору, организующей прохождение через ЭВМ потока задач, распределяющей между ними ресурсы, обнаруживающей ошибки в программах пользователя и сбои и неисправности в оборудовании. Такая программа, а в современных ЭВМ даже целый комплекс программ, выполняющих перечисленные функции, является основной частью математического или программного обеспечения вычислительной системы (ВС). К математическому обеспечению относятся также трансляторы с языков программирования, средства хранения и коррекции программ и наборов данных, средства отладки программ, построения из отдельных программ больших программных систем, а также уже готовые специализированные комплексы прикладных программ, обслуживающие коллективы пользователей конкретной ВС.

Весь такой комплекс программ, построенный по некоторому плану и способный согласованно работать, получил наименование *операционной системы* (ОС). Значительную часть ОС с точки зрения пользователя можно считать непосредственным продолжением аппаратной части ЭВМ, которую пока нецелесообразно или невозможно реализовать аппаратными средствами.

Операционные системы приняли на себя функцию общения программиста с машиной. Как при пакетном, так и при монопольном режиме использования машины

программист вынужден работать в рамках, определяемых ОС. Свои требования он может излагать только на ее входном языке.

Различные операционные системы имеют и различные входные языки (иногда несовместимые, как это имеет место для машинных языков ЭВМ), но наличие некоторой известной программисту ОС позволяет ему не принимать во внимание марку ЭВМ и ее машинный язык. Таким образом ОС создает некоторую вычислительную среду, которую только и должен учитывать программист. Заботы о несовместимости программ, составленных с помощью различных версий трансляторов, в современных ОС отпадают. Важна *версия* операционной системы, причем процессы ее доработки обычно ведутся так, чтобы сохранить совместимость этой версии с предыдущими.

Программист пишет программу на одном из алгоритмических языков, допускаемых ОС, и сопровождает ее набором операторов управления заданиями на входном языке этой операционной системы. После этого программа может быть транслирована в коды ЭВМ и выполнена на любой ЭВМ с той же операционной системой.

В дальнейшем изучение приемов работы программиста на ЭВМ, управляемой ОС, мы будем вести на примере ДОС/ЕС — *дисковой операционной системы,* которая обеспечивает решение задач на всех машинах Единой Системы ЭВМ, созданной совместно странами — членами СЭВ. Полное изложение возможностей этой операционной системы выходит за рамки данной книги. Краткое описание ДОС/ЕС дается в гл. 1 книги.

Комплекс программ УПРАВЛЕНИЕ ЗАДАНИЯМИ рассмотрен в гл. 2. В частности, в этой главе описаны средства программиста — операторы УПРАВЛЕНИЯ ЗАДАНИЯМИ и средства оператора — директивы УПРАВЛЕНИЯ ЗАДАНИЯМИ (эту главу рекомендуется изучить наиболее подробно, так как реализуемый с помощью описанных в ней средств пакетный режим работы ЭВМ для ДОС/ЕС является основным).

Гл. 3 книги содержит описание включенных во все версии ДОС/ЕС языков: базисного ФОРТРАНа и ПЛ/1. Изложение этих языков ведется путем сравнения их со знакомым большинству программистов АЛГОЛом.

Гл. 4 посвящена описанию таких конструкций языков программирования, как операторы ввода и вывода, объ-

явления, открытия и закрытия файлов и операторы управления данными.

БИБЛИОТЕКАРЬ — комплекс программ ДОС/ЕС, позволяющих управлять хранением, изменением, перемещением и удалением из памяти программ на различных этапах их обработки, описан в гл. 5.

Использование программы РЕДАКТОР, позволяющей строить комплексные программы из отдельных частей — модулей, описывается в гл. 6.

Способы хранения, перемещения и удаления наборов данных, а также средства их защиты от разрушения и несанкционированного доступа описаны в гл. 7.

Все главы содержат примеры, иллюстрирующие использование описываемых средств.

Автор пользуется случаем выразить свою благодарность проф. А. И. Китову, к. т. н. Л. Д. Райкову и к. т. н. С. Н. Бушеву, просмотревшим текст книги и сделавшим ряд замечаний, способствовавших его улучшению, а также проф. К. А. Пупкову, взявшему на себя труд по ее редактированию.

1. Дисковая операционная система Единой Системы ЭВМ

1.1. Единая Система ЭВМ

Единая Система ЭВМ создана коллективными усилиями стран социалистического содружества—Болгарии, Венгрии, ГДР, Польши, СССР и Чехословакии. Система состоит из ряда ЭВМ производительностью от нескольких десятков до миллиона и выше операций в секунду. Все ЭВМ построены по аналогичным схемам и являются программно-совместимыми. Обеспечена программная совместимость и с рядом зарубежных ЭВМ.

Периферийные устройства ЕС ЭВМ снабжены средствами стандартного сопряжения, что позволяет вводить их в состав любой вычислительной установки с процессором ЕС ЭВМ. Программная совместимость позволяет не только использовать ранее разработанные программы, но и постепенно наращивать мощность вычислительных средств по мере освоения пользователями возможностей машины и расширения круга решаемых задач.

Все модели ЕС ЭВМ работают под управлением операционных систем, поставляемых пользователю одновременно с ЭВМ. В настоящее время пользователю поставляется одна из следующих двух операционных систем:

— дисковая операционная система ЕС ЭВМ (ДОС/ЕС), применение которой рекомендуется для моделей ЕС ЭВМ с оперативной памятью 64—128К байтов и относительно малой производительностью;

— операционная система ЕС ЭВМ (ОС/ЕС) — для моделей ЕС ЭВМ с оперативной памятью 128—256К и более байтов.

Следует заметить, что при наличии нужного объема оперативной памяти ОС/ЕС может функционировать и на самых младших моделях ЕС ЭВМ, но ее использование на таких моделях по ряду причин нерационально.

В ЕС ЭВМ как сами машины, так и их операционные системы постоянно совершенствуются. Для опера-

ционных систем это отмечается присвоением конкретным реализациям номера версии и модификации. Так, в частности, пользователь может встретить на какой-либо ЭВМ операционную систему ДОС/ЕС версии 1.3, 2.0 или 2.2 или операционную систему ОС/ЕС версии 1.0, 4.1 и т. д. Это развитие происходит таким образом, что все программы, составленные и отлаженные на более ранних версиях операционных систем, остаются пригодными при работе ЭВМ под управлением более поздних версий ДОС/ЕС или ОС/ЕС.

1.2. Структура и состав ДОС/ЕС

ДОС/ЕС предназначена для более полной загрузки машины и повышения производительности труда программиста; она, в частности, позволяет:

— делить задачу на части, программировать и отлаживать эти части отдельно и далее объединять их для решения исходной задачи;

— использовать разработанную программу как непосредственно после ее создания, так и в другое время, для чего в составе ДОС/ЕС имеются средства хранения и поиска программ;

— делить большие программы, которые не могут быть размещены в оперативной памяти целиком, на сегменты, называемые фазами; эти фазы могут вызываться в оперативную память в необходимом порядке;

— писать программы, которые не зависят от количества или конкретного набора периферийных устройств, указывая в тексте только имена так называемых *логических* устройств и привязывая их к конкретным, так называемым *физическим* устройствам непосредственно перед выполнением программы.

Состав ДОС/ЕС и ее структура показаны на рис. 1.1. Дисковая операционная система ЕС ЭВМ способна настраиваться на заданные состав средств вычислительной системы, круг задач и режимы работы. Процесс этой настройки называется *генерацией*.

В результате генерации ДОС/ЕС может обмениваться информацией с определенным количеством периферийных устройств заданных типов; транслировать программы с языков заданного перечня; осуществлять без помощи программиста некоторый набор действий и предоставлять пользователю некоторый комплекс услуг (режимов).

Не следует считать, что при генерации системы все возможности выгодно выбирать по максимуму. Реализация каждой дополнительной возможности или услуги, даже если ею не пользуются, требует определенных затрат памяти и какого-то уменьшения реального быстро-

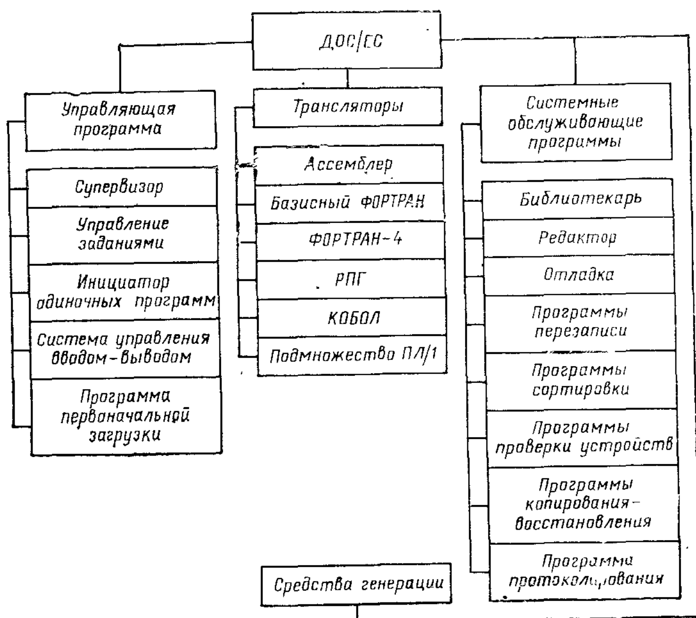


Рис. 1.1. Структура и состав ДОС/ЕС.

действия машины. Поэтому при выборе характеристик ДОС на этапе ее генерации необходимо учитывать реальные потребности пользователей, а не какие-либо редкие сочетания требований к материальным, языковым и программным средствам вычислительной системы.

Режимы и услуги, предусмотренные при генерации, называются *стандартными* и действуют в ДОС с момента ее первоначальной загрузки, т. е. с момента ввода ее в оперативную память ЭВМ со специального пакета дисков. Отклонения от стандартных режимов, если они потребуются, могут быть осуществлены с помощью специальных указаний оператора или программиста. Однако указания программиста теряют силу после окончания

работы ДОС над его заданием, а указания оператора остаются в силе до тех пор, пока не понадобится повторить первоначальную загрузку системы.

Процесс обработки информации в вычислительной системе делится на части, именуемые *заданиями*. Отличительная особенность задания — его независимость от других заданий. Выполнение или невыполнение какого-либо задания не влияет на возможность выполнения последующих заданий.

Задание может подразделяться на *шаги* задания или *задачи*, которые состоят в выполнении какой-либо программы, обрабатывающей определенную порцию данных. Так, выполнение одной и той же программы с различными порциями данных считается выполнением различных задач.

Шаги задания могут зависеть друг от друга. Классическим примером трехшагового задания является трансляция исходной программы, ее редактирование и, наконец, выполнение. Обнаружение ошибок в исходной программе, естественно, отменяет выполнение последующих шагов задания.

Данные, которые ДОС/ЕС извлекает из внешней памяти или размещает в этой памяти, а также данные, выдаваемые на АЦПУ, именуются *файлами* или *наборами данных*. Файлы состоят из отдельных логически законченных порций — *записей*.

Файл как логически определенное множество записей может размещаться на одной физической единице внешней памяти — бобине магнитной ленты, пакете дисков или на нескольких таких единицах, именуемых *томами*. Соответственно различают *однотомные* и *многотомные* файлы. Файлы малого объема объединяются в *многофайловые* томы.

Так как на магнитных лентах (и в меньшей степени на дисках) промежутки между отдельными записями достаточно велики, то короткие записи оказываются невыгодными и их объединяют в группы, называемые *блоками*. При этом иногда записи называют *логическими записями*, а блоки — *физическими записями*.

Решение задач на ЕС ЭВМ можно выполнять как в *однопрограммном*, так и в *многопрограммном* режиме (режиме *мультипрограммирования*). На машинах, управляемых дисковой операционной системой, мультипрограммирование реализовано в простейшем своем вари-

анте — с возможностью решения фиксированного числа задач. Этот режим, называемый *мультипрограммированием с фиксированным числом задач*, обозначается сокращенно MFT. Он предполагает разбиение оперативной памяти и периферийных устройств на так называемые *разделы*, предназначаемые для каждой из решаемых задач.

В ДОС/ЕС допускается создание не более трех таких разделов: фонового раздела (BG) и двух разделов переднего плана (F2, F1).

Возможность эксплуатации ДОС в режиме мультипрограммирования должна быть предусмотрена на этапе генерации системы, но разбиение перечисленных ресурсов машины на разделы можно изменять в процессе первоначальной загрузки, а распределение периферийных устройств — даже в процессе выполнения отдельных шагов задания.

Ниже всюду для сокращения вместо термина «периферийные устройства» мы будем употреблять термин «устройства».

1.3. Физические и логические устройства

Физические устройства адресуются номером канала, который их обслуживает, и порядковым номером устройства, присоединенного к данному каналу. Номера эти записываются в 16-ричной системе счисления и кодируются для ЕС ЭВМ в виде X'cuu', где X — признак 16-ричной системы счисления, c — номер канала (для ЕС ЭВМ $0 \leqslant c \leqslant 6$, причем нулевым номером обозначается мультиплексный канал, а остальными — селекторные), а uu — номер устройства. Всего к каналу подключается до 254 устройств с адресами $0 \leqslant uu \leqslant$ X'FE'.

Логические устройства в ДОС/ЕС делятся на *системные устройства* и *устройства программиста*[1] и кодируются именами SYSxxx. Под системными устройствами понимаются те из них, которые используются системой для собственных надобностей. Для системных

---

[1] **Шура-Бура М. Р.** и др. Операционная система ДОС/ЕС. М., «Статистика», 1975 (с. 11).

устройств вместо символов xxx ставятся прописные латинские буквы, сокращенно описывающие их назначение, а для устройств программиста — трехзначные десятичные числа.

К системным устройствам относятся:

SYSRES для размещения основных компонентов ДОС/ЕС; это устройство называется *резиденцией* системы, размещенный на нем пакет дисков — *резидентным пакетом*, а входящий в состав резидентного пакета текст ДОС/ЕС — *резидентным файлом*;

SYSRDR для ввода управляющих операторов задания;

SYSIPT для ввода наборов данных, а иногда и управляющих операторов. Это устройство называется *системным вводом*;

SYSPCH для выдачи данных в формате перфокарт. Так как в качестве этого устройства чаще всего используют перфоратор, то оно называется *системным перфоратором*;

SYSLST для выдачи материалов, подготовленных к печати (*системная печать*);

SYSLOG — пишущая машинка оператора;

SYSLNK для накопления входной информации программы — РЕДАКТОРА.

Резидентный файл состоит из трех *системных библиотек* ДОС/ЕС:

— библиотеки *абсолютных модулей* (CL);

— библиотеки *объектных модулей* (RL);

— библиотеки *исходных модулей* (SL).

Но ДОС/ЕС позволяет расширить библиотечное хозяйство за счет создания так называемых личных, а по существу дополнительных библиотек.

Для размещения личных библиотек CL, RL и SL в составе ДОС/ЕС предусмотрены три логических устройства SYSCLB, SYSRLB и SYSSLB соответственно.

С ДОС/ЕС взаимодействуют следующие физические устройства ЕС ЭВМ:

— устройство ввода с перфокарт ЕС-6012 или ЕС-6016;

— устройство вывода на перфокарты ЕС-7010 или ЕС-7012;

— устройство выдачи на печать (АЦПУ) ЕС-7030, ЕС-7031, ЕС-7032, ЕС-7033 или ЕС-7035;

— пишущая машинка оператора ЕС-7070, ЕС-7071, ЕС-7072, ЕС-7073 или ЕС-7074;

— накопитель на магнитных дисках ЕС-5050, ЕС-5052, ЕС-5056 или ЕС-5061;

— накопитель на магнитных лентах ЕС-5010, ЕС-5012, ЕС-5016, ЕС-5017, ЕС-5019 или ЕС-5021.

Другие устройства, выпускаемые в стандартных конфигурациях ЕС ЭВМ, например устройства ввода — вывода на перфоленту, а также графопостроители, дисплеи и другое оборудование ЕС ЭВМ, могут быть сделаны доступными дисковой операционной системе ЕС путем написания специальных программ на физическом уровне.

1.4. Аппарат назначений

Как уже говорилось, ДОС/ЕС позволяет программисту при написании программ не указывать конкретное устройство или даже тип устройства, на которое предполагается послать или с которого считать информацию. В программе можно ограничиться только указанием имен используемых логических устройств. Установление же соответствия между логическими и физическими устройствами может целиком выполняться за пределами его программы с помощью аппарата назначений. Но для программиста, желающего заниматься этим самостоятельно, ДОС/ЕС предоставляет и такую возможность.

Аппарат назначений ДОС/ЕС позволяет использовать три вида назначения логических устройств физическим: стандартные, постоянные и временные.

Стандартными, как говорилось выше, называются назначения, которые производятся на этапе генерации системы. Они могут отменяться оператором и программистом и заменяться другими назначениями, но в процессе первоначальной загрузки системы они восстанавливаются.

Постоянные назначения делает оператор со своего пульта, вводя соответствующие директивы программы УПРАВЛЕНИЕ ЗАДАНИЯМИ (УЗ). Эти назначения могут отменяться программистом, но его отмена дейст-

вует только до конца задания, т. е. до считывания системой оператора УЗ вида:

/&

Таблица 1.1

Логические устройства	Функции	Типы физических устройств					
		Ввод перфокарт	Вывод перфокарт	Пультовая машинка	АЦПУ	НМД	НМЛ
SYSRES	Размещение ДОС					+	
SYSRDR	Ввод управляющих операторов задания	+				+	+
SYSIPT	Системный ввод потока данных и некоторых операторов	+				+	+
SYSPCH	Системный вывод		+			+	+
SYSLST	Системиая печать				+	+	+
SYSLOG	Связь ДОС и оператора	•		+			
SYSLNK	Накопление данных для РЕДАКТОРА					+	
SYSREG	Регистрация статистической информации					+	
SYSCLB	Размещение личиой библиотеки абсолютных модулей					+	
SYSRLB	Размещение личной бнблиотекн объектных модулей					+	
SYSSLB	Размещение личной библиотеки исходных модулей					+	
SYSnnn	Используется по усмотрению программиста	+	+		+	+	+

после чего вступают в силу постоянные назначения, а если их не было, то стандартные назначения.

Временные назначения программист делает, вводя в состав задания операторы УЗ. Временные назначения имеет возможность делать и оператор, если ему нужно, чтобы они потеряли силу после окончания текущего задания.

При назначениях необходимо учитывать следующие ограничения:

— разделы переднего плана и фоновой раздел должны иметь разные назначения, кроме устройства SYSLOG, в качестве которого во всех трех разделах используется, как правило, одна общая пультовая пишущая машинка оператора;

— неперекрывающиеся участки памяти на магнитных дисках могут быть назначены различным логическим устройствам одного или разных разделов;

— в качестве устройств SYSRDR и SYSIPT можно использовать одно общее физическое устройство (это очень удобно и часто так и поступают);

— назначения для системных устройств SYSRDR, SYSIPT, SYSLST и SYSPCH на диски не могут быть временными.

Для определенных логических устройств имеются ограничения также по типам назначаемых им физических устройств. Эти ограничения приведены в табл. 1.1 (знак +). Звездочка в табл. 1.1 означает возможность использования устройства при аварийном режиме работы машины.

1.5. СУПЕРВИЗОР

СУПЕРВИЗОР — основная составная часть управляющей программы ДОС/ЕС. Это единственная из программ системы, которой разрешено пользоваться привилегированными операциями. СУПЕРВИЗОР управляет ходом выполнения любой из задач, с ввода задания до получения результатов. От выполняемой программы управление передается СУПЕРВИЗОРУ по сигналу прерывания, а после обработки прерывания СУПЕРВИЗОР определяет программу, которой он и передает управление дальнейшим ходом процесса.

СУПЕРВИЗОР состоит из нескольких фаз, хранящихся в CL. Только корневая его фаза — ядро — всегда находится в оперативной памяти. Остальные фазы способны сами настраиваться на адрес их размещения в оперативной памяти и поэтому называются *самоперемещающимися* или *транзитами*. Они вызываются для выполнения в так называемую транзитную область оперативной памяти супервизора. Вызов транзитов осуществляется ядром, ядро же вызывается программой первоначальной загрузки ДОС/ЕС.

Состав ядра СУПЕРВИЗОРА переменный; он определяется потребителем при генерации. В частности, на состав ядра влияет:

— необходимость обеспечения многопрограммной работы;

— возможность пакетной обработки в разделах переднего плана;

— состав периферийных устройств и распределение их между разделами.

На рис. 1.2 дана примерная структура участка памяти, занимаемого СУПЕРВИЗОРОМ.

Постоянно распределенные ячейки	
Область связи	
Ядро	Распознавание прерываний Планировщик каналов Системный загрузчик Устранение ошибок Защита памяти Обслуживание таймера Таблицы СУПЕРВИЗОРА
Транзитная область	

Рис. 1.2. Структура памяти, занимаемой СУПЕРВИЗОРОМ.

Программист, работающий на языке АССЕМБЛЕРА, может передать управление СУПЕРВИЗОРУ с помощью команды SVC. Это оказывается необходимым при использовании физической системы управления вводом — выводом. Но большей частью, даже на языке АССЕМБЛЕРА, программист пользуется *макрокомандами,* т. е. обращается к готовым подпрограммам, и у него нет необходимости непосредственно обращаться к СУПЕРВИЗОРУ. При написании же программ на языках высокого уровня прямые обращения к СУПЕРВИЗОРУ невозможны.

Языки высокого уровня, давая программисту большие удобства в составлении программ, вместе с тем не позволяют использовать некоторые не предусмотренные при составлении трансляторов возможности. Поэтому даже при работе с таким универсальным языком, как ПЛ/1, иногда приходится отдельные части алгоритма программировать на языке АССЕМБЛЕРА, давая возможность применять машинную систему команд в полном объеме.

Первичная передача управления СУПЕРВИЗОРУ осуществляется *программой первоначальной загрузки.* Во всех остальных случаях СУПЕРВИЗОР получает управление в результате *прерываний.*

1.6. Прерывания и их виды

Прерыванием называется прекращение выполнения текущей программы под воздействием некоторых ситуаций в процессоре и устройствах ввода — вывода (УВВ) — *причин прерывания.* В ЕС ЭВМ имеется пять групп причин прерывания:

— прерывания ввода — вывода;
— обращение к СУПЕРВИЗОРУ;
— программные сбои;
— машинные сбои;
— внешние прерывания.

При возникновении прерывания СУПЕРВИЗОР запоминает состояние прерванной программы: содержимое всех регистров, слово состояния прерванной программы («старое» PSW), имя этой программы и другую информацию. Запоминание производится в так называемой *области сохранения*, которая для фонового раздела находится в области памяти, выделенной СУПЕРВИЗОРУ, а для других разделов в начале их участков памяти.

Структура области сохранения показана на рис. 1.3.

<table>
<tr><td colspan="2">Имя программы
(8 байтов)</td><td>Старое PSW
(8 байтов)</td></tr>
<tr><td colspan="3">Общие регистры
(64 байта)</td></tr>
<tr><td>Длина области меток
(2 байта)</td><td>Резерв
(2 байта)</td><td>Время начала работы над заданием
(4 байта)</td></tr>
<tr><td colspan="3">Регистры плавающей точки
(32 байта)</td></tr>
</table>

Рис. 1.3. Структура области сохранения.

После заполнения *области сохранения* СУПЕРВИЗОР уточняет причину прерывания и в соответствии с найденной причиной подключает одну или несколько *программ обработки прерываний* этого типа.

Группа прерываний *обращение к СУПЕРВИЗОРУ* (команда SVC) применяется при необходимости:

— выполнить операцию ввода — вывода;
— открыть или закрыть файл;
— вызвать очередную фазу многофазной программы;
— сообщить об окончании выполнения программы.

Как среди макрокоманд языка АССЕМБЛЕРА, так и среди операторов алгоритмических языков высокого уровня имеются возможности описания этих работ.

По прерыванию от *программного сбоя* СУПЕРВИЗОР передает управление подпрограмме обработки сбоев программы пользователя, если ее наличие и имя сообщено СУПЕРВИЗОРУ. В противном случае СУПЕРВИЗОР прекращает выполнение задания, выдавая по требованию программиста на SYSLST содержимое основной памяти раздела, в котором решается задача.

Прерывания по *машинным сбоям* вызывают, как правило, перевод СУПЕРВИЗОРОМ ДОС в состояние ожидания и выдачу сигнала о необходимости повторения процедуры первоначальной загрузки ДОС.

Прерывания ввода — вывода имеют место обычно при освобождении канала, группового устройства управления или конкретного УВВ. Обработка этих прерываний осуществляется специальным блоком СУПЕРВИЗОРА — *планировщиком каналов.*

К *внешним* относятся прерывания от линий связи с удаленными абонентами, от других ЭВМ в многомашинных комплексах, от таймера и от кнопки ВНИМАНИЕ пультовой пишущей машинки оператора. СУПЕРВИЗОР их игнорирует, за исключением прерывания от кнопки ВНИМАНИЕ и прерывания от таймера, если при генерации системы предусмотрено его использование.

1.7. Планировщик каналов

Планировщик каналов является частью СУПЕРВИЗОРА, содержащей несколько программ, обслуживающих операции ввода — вывода.

Функции планировщика каналов иллюстрируются рис. 1.4.

Для понимания этих функций необходимо рассмотреть способы подготовки операций ввода — вывода в ДОС/ЕС.

Для функционирования УВВ необходимо сначала составить программу канала и задать адресное слово канала (заполнить специально выделенные 8 байтов памяти), далее необходимо сформировать программу управления каналами, проанализировать результаты запуска УВВ, а также обработать прерывания, поступающие от УВВ. Часть этих операций может быть выполнена только СУПЕРВИЗОРОМ, так как они требу-

ют использования привилегированных команд. Остальные операции выполняются либо программами пользователя, либо системными программами, вызываемыми программой пользователя. Соответственно этому различают *физический* и *логический* уровень программирования процессов ввода — вывода. Каждый из них использует свою совокупность программ ДОС — физическую систему управления вводом — выводом (ФСУВВ) и логическую систему управления вводом — выводом (ЛСУВВ).

На *физическом уровне* программист готовит: программу канала; блок управления командами канала (блок ССВ) с необходимой информацией для СУПЕРВИЗОРА; запрос СУПЕРВИЗОРУ с помощью команды SVC или макрокоманды EXCP на языке АССЕМБЛЕРА, а также программу обработки результатов выполнения операции ввода — вывода.

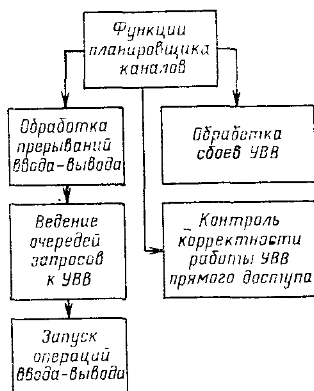


Рис. 1.4. Функции планировщика каналов.

Макрокоманда ССВ строит блок ССВ в программе пользователя. Этот блок служит программе средством обмена информацией с СУПЕРВИЗОРОМ.

Макрокоманда ССВ имеет формат

[<имя блока ССВ>] ССВ SYSnnn,<адрес CCW>[,X'nnnn'] [,<адрес уточненного состояния>]

В программе канала кроме уже упомянутых макрокоманд используется еще макрокоманда WAIT (ждать). Во всех макрокомандах имя блока ССВ должно быть одинаковым. Операнд SYSnnn обозначает логическое УВВ, участвующее в операции ввода — вывода. Операнд <адрес ССВ> должен указывать место расположения первой команды программы канала. Остальные операнды необязательны.

Макрокоманда EXCP обеспечивает выполнение программы канала в соответствии с содержимым блока ССВ. Ее формат

[<имя>] EXCP<имя блока ССВ>

Макрокоманда WAIT применяется, когда продолжение программы пользователя невозможно без окончания операции ввода — вывода. Она имеет вид

[<имя>] WAIT<имя блока CCB>

Как указывалось выше, имя блока CCB как операнд макрокоманд EXCP и WATT должно соответствовать имени, стоящему перед макрокомандой CCB.

Пользуясь *логическим уровнем* программирования ввода — вывода, программист может заботиться главным образом об описании *структуры данных*. Например, на ФОРТРАНЕ оператор ввода информации в память имеет вид

READ (a, b) <список>

Здесь a — «номер файла», т. е. фактически условное цифровое наименование логического УВВ, участвующего в обмене; b — метка оператора FORMAT, описывающего структуру данных; <список> — перечень идентификаторов переменных или массивов, которые передаются в память.

Выбирая различные значения a и делая необходимые назначения, мы можем вводить данные с перфокарт, дисков, лент и т. п.

Функции планировщика каналов рассмотрим, начиная с процедуры ведения очередей запросов к УВВ. Планировщик каналов, выполняя эту функцию, выбирает из CCB наименование требуемого логического УВВ и выясняет номер назначенного ему физического УВВ, пользуясь таблицами СУПЕРВИЗОРА (см. ниже § 1.10). После этого запрос ставится в очередь к этому устройству.

Далее анализируется состояние запрошенного УВВ, и если оно занято, а программа пользователя исчерпала возможности работать без реализации запроса, то она переводится в состояние ожидания, а центральный процессор переходит к выполнению другой программы, пока операция ввода — вывода не будет выполнена.

При освобождении какого-либо УВВ СУПЕРВИЗОР анализирует очередь к нему и по CCB очередного запроса определяет характеристики нужного УВВ, формирует адресное слово канала и выдает команду НАЧАТЬ ВВОД — ВЫВОД. При этом запрос исключается из очереди.

Прерывания от устройства SYSLOG возникают при нажатии оператором кнопки ВНИМАНИЕ. Планиров-

щик каналов рассматривает их как требования установить связь с оператором.

Процесс обработки сбоев ввода — вывода основывается на анализе содержимого так называемого *слова состояния канала* (CSW) — участка памяти длиной в 8 байтов, формируемого машиной в момент прерывания (рис. 1.5).

Номер байта	Номера битов							
	0	1	2	3	4	5	6	7
0	Ключ защиты				0	0	0	0
1 2 3	Адрес команды канала							
4	Состояние устройств							
5	Состояние канала							
6 7	Счетчик команд канала							

Рис. 1.5. Слово состояния канала.

В CSW имеются разряды, отведенные для регистрации сбоев. Обнаружив наличие в них единиц, СУПЕРВИЗОР передает управление программе обработки сбоев и в зависимости от обстоятельств может:

— попытаться избавиться от сбоя(например, потребовав от оператора замены поврежденной карты);

— игнорировать сбой;

— передать управление для принятия решения программе пользователя (если в ней предусмотрены для этого специальные алгоритмы);

— снять задание.

Последнее решение принимается в том случае, когда ни избавиться от сбоя, ни поручить разобраться с ним программе пользователя не оказалось возможным, а по своей серьезности его нельзя игнорировать.

1.8. Системный загрузчик

Функцию загрузки, т. е. *вызова программ* в оперативную память для выполнения обеспечивает часть ядра СУПЕРВИЗОРА — *системный загрузчик* (СЗ). Единственная программа, загружаемая помимо системного загрузчика,— само ядро СУПЕРВИЗОРА. Ее, как сказано выше, загружает программа первоначальной загрузки.

Большинство фаз в CL построено так, что их необходимо размещать во вполне определенных местах оперативной памяти. Но имеются и самоперемещающиеся фазы, которые можно загрузить в любое место памяти. Таковы, например, транзиты СУПЕРВИЗОРА.

СЗ загружает в память как фазы программы пользователей, так и все фазы самой ДОС/ЕС.

При завершении любой программы посылается сигнал об этом СУПЕРВИЗОРУ. При составлении программ на языке ассемблера программист использует макрокоманду EOJ. В программах на алгоритмических языках эту макрокоманду формируют трансляторы.

При *нормальном* завершении шага задания СУПЕРВИЗОР организует выполнение следующего шага. В противном случае печатается *сообщение о причинах прекращения* шага задания и выполнение всех последующих шагов задания отменяется. В режиме DUMP (см. ниже гл. 2) делается распечатка содержимого соответствующего раздела основной памяти, чтобы программист, анализируя ее, разобрался в причинах ошибок. Далее СУПЕРВИЗОР вызывает на место прерванного задания программу УПРАВЛЕНИЕ ЗАДАНИЯМИ для подготовки следующего задания.

1.9. Связь оператора с ДОС/ЕС

Связь оператора с СУПЕРВИЗОРОМ является частью его связи с ДОС/ЕС в целом. Назначение этой связи — получать информацию о работе ДОС и передавать ей свои директивы.

Передача информации от ДОС к оператору идет через SYSLOG. Сообщения, поступающие от ДОС, делятся на информационные сообщения и запросы, требующие от оператора тех или иных действий. Структура сообщения ДОС:

<сообщение ДОС> : : =<код сообщения> <индикатор реакции оператора><пробел><текст>

<код сообщения> : :=<автор сообщения> <номер сообщения>

<индикатор реакции оператора> : : =<символ>

<текст> : : = <произвольной длины строка>

<автор сообщения> : : =<16-ричная цифра>

<номер сообщения> : : =<три символа>

Индикатор реакции оператора — буквенный символ:

— символ A требует от оператора вполне определенного действия;

— символ D требует выбора одного из рекомендуемых действий;

— символ I действий не требует;

— если же в сообщении появились символы W или S, то ДОС стала неработоспособной и необходимо повторить процесс первоначальной загрузки.

Символ <автор сообщения> для различных компонентов ДОС/ЕС имеет значение:

— СУПЕРВИЗОР	0
— программа УПРАВЛЕНИЕ ЗАДАНИЯМИ	1
— РЕДАКТОР	2
— БИБЛИОТЕКАРЬ	3
— логическая система управления вводом — выводом	4
— транслятор с языка ПЛ/1	5
— транслятор с языка РПГ	6
— программы сортировки	7
— программы перезаписи	8
— программа отладки	9
— АССЕМБЛЕР	A
— транслятор с языка ФОРТРАН	B

Номер сообщения — обычно трехзначное десятичное число или три символа, вполне определяющие содержание сообщения.

Текст только разъясняет сообщение *оператору*, если оно выдается на SYSLOG, и *программисту* при выдаче на SYSLST.

Если при генерации ДОС был предусмотрен мультипрограммный режим, то перед описанными сообщениями через пробел ставится код раздела, к которому он относится:

<код раздела>::=BG|F1|F2|AR|SP

где AR — код программы ВНИМАНИЕ, а SP — код сообщений, не относящихся конкретно ни к одному из разделов. Многие из этих сообщений, будучи напечатаны на SYSLST, могут быть адресованы и программисту.

Сообщения оператора в адрес ДОС — это либо директивы, либо ответы на запросы ДОС.

Назначение директив оператора состоит в том, чтобы узнать состояние системы, управлять работой ДОС и использованием УВВ. Оператор может посылать дирек-

тивы тем программам, которые способны их воспринимать, т. е. имеют подпрограммы принятия и обработки директив.

Программу, воспринимающую директиву, будем называть *адресатом директивы.* Из системных программ адресатами директив могут быть: программа ВНИМАНИЕ, программа УПРАВЛЕНИЕ ЗАДАНИЯМИ (УЗ) и ИНИЦИАТОР ОДИНОЧНЫХ ПРОГРАММ (ИОП). Шифры этих адресатов: AR, JC, SPI.

В ДОС/ЕС принят следующий порядок ввода директив:

— нажимается кнопка ВНИМАНИЕ, что вызывает прерывание ввода — вывода и передачу управления СУПЕРВИЗОРУ. Распознав это прерывание, СУПЕРВИЗОР вызывает один из своих *транзитов* — программу ВНИМАНИЕ, которая печатает на SYSLOG:

READY FOR COMMUNICATION

— после получения этого сообщения оператор может вводить с помощью пультовой пишущей машинки тексты директив (для программ УЗ и ИОП возможен ввод директив и на перфокартах через SYSRDR);

— для прекращения связи вводится директива КТ (нажимается кнопка КТ).

В ДОС/ЕС возможна также *связь оператора с программами пользователей,* если это предусмотрено при генерации системы.

Для связи с программой УПРАВЛЕНИЕ ЗАДАНИЯМИ (что требуется наиболее часто) оператор может использовать директивы:

BATCH (начать пакетную обработку);

STOP (временно остановить пакетную обработку);

START (возобновить временно прекращенную пакетную обработку);

PAUSE (приостановить ввод управляющих операторов и настроиться на связь с человеком-оператором);

UNBATCH (прекратить пакетную обработку в одном из разделов переднего плана);

MAP (напечатать таблицу распределения памяти) и ряд других.

Некоторые из этих директив являются также директивами программы ВНИМАНИЕ.

Связь с *инициатором одиночных программ* может быть установлена директивой START, если только в этом разделе нет пакетной обработки.

Пример 1. Определить распределение памяти между разделами и выполняемые в них задания.

Решение.

— устанавливаем связь с ДОС, т. е. нажимаем кнопку ВНИМАНИЕ и после ответа системы вводим директиву MAP.

Пусть по этой директиве система на SYSLOG напечатала таблицу

SP		8	16 383	
BG	T	10	36 863	TSTR7
F2		4	45 055	
F1		10	65 535	CDTR

Символ T во второй колонке показывает, что таймер закреплен за фоновым разделом. В третьей колонке приведены размеры разделов в блоках, кратных 2К (2048 байтов). Таким образом, область, занимаемая управляющей программой ДОС, составляет 16К байт, фоновый раздел — 20К байт и т. д. В четвертой колонке показаны верхние границы разделов, т. е. номера последних байтов соответствующих областей памяти. Наконец, в последней колонке напечатаны имена выполняемых заданий: в разделе BG выполняется задание TSTR7, в F1—CDTR (программа перезаписи КАРТЫ — ЛЕНТА), а раздел F2 свободен. Получив таблицу, нажимаем кнопку КТ, чем возвращаем управление программе УПРАВЛЕНИЕ ЗАДАНИЯМИ.

Пример 2. Прекратить выполнение CDTR в F1 и обеспечить возможность выполнения этой программы в F2 (для нее требуется не менее 10К байтов памяти).

Решение.

— устанавливаем связь с системой;

— снимаем задание в F1 (раздел F1 освобождается;)

CANCEL F1

— так как в разделе BG идет выполнение задания (раздел активен), то уменьшать размер его области нельзя; поэтому перераспределяем память между F1 и F2:

ALLOC F1=4K, F2=10K

— делаем нужные назначения УВВ для разделов F1 и F2 (см. ниже в разделе УПРАВЛЕНИЕ ЗАДАНИЯМИ);

— запускаем пакетную обработку в F2:

BATCH F2

— возвращаем управление программе УЗ.

Пример 3. Снять задание в F2 и организовать в этом разделе режиме выполнения одиночной программы.

Решение.

— устанавливаем связь с ДОС;

— снимаем задание в F2 и отменяем в нем пакетный режим:

CANCEL F2

UNBATCH F2

— сделав все необходимые назначения УВВ, запускаем в F2 одиночную программу

START F2

Выполнение этой директивы передает управление программе ИНИЦИАТОР ОДИНОЧНЫХ ПРОГРАММ[1], которая в книге не описывается.

Другие возможности связи оператора с системой будут обсуждены ниже, при рассмотрении структуры и функций программы УЗ.

1.10. Таблицы СУПЕРВИЗОРА

Как видно из рис. 1.2, СУПЕРВИЗОР хранит в оперативной памяти таблицы. Бóльшая часть этих таблиц (такие, как таблицы логических и физических УВВ, состояния программ, состояния системных файлов, очередей к УВВ, устройств, в которых имели место сбои, и и др.) программистом не используется.

Но одна из таблиц является общей для СУПЕРВИЗОРА и некоторых программ пользователей. Это так называемая *область связи*, с помощью которой можно управлять использованием меток в программах перезаписи, а также локализовать ошибки периода работы программы на языке ПЛ/1.

При мультипрограммном режиме ДОС/ЕС для каждого из разделов создает свою область связи, но все эти области располагаются внутри области, отведенной управляющей программе. Программы пользователя могут иметь доступ к области связи своего раздела только для считывания. От записи область связи защищена и информацию, содержащуюся в ней, может изменять только СУПЕРВИЗОР.

Ниже показана структура области связи (рис. 1.6).

---

[1] Программа ИНИЦИАТОР ОДИНОЧНЫХ ПРОГРАММ, в отличие от программы УПРАВЛЕНИЕ ЗАДАНИЯМИ, требует не 10К, а 2К байтов оперативной памяти. Так как модели ЕС ЭВМ в настоящее время выпускаются с памятью 128К и более байтов, то в настоящей книге программа ИНИЦИАТОР ОДИНОЧНЫХ ПРОГРАММ не описывается.

Номера байтов	Содержимое	Номера байтов	Содержимое
0 1 2 3 4 5 6 7	Календарная дата	23	Переключатели
8 9	Адрес начала программы пользователя	24 25 26 27 28 29 30 31	Имя задания
10 11	Адрес начала области защиты ключом 1	32 33 34 35	Адрес конца раздела
12 13 14 15 16 17 18 19 20 21 22	Область потребителя для внутрипрограммных и межпрограммных связей	36 37 38 39	Адрес конца фазы
		40 41 42 43	Адрес конца программы
		44 45	Длина области меток

Рис. 1.6. Структура области связи.

2. Управление заданиями

2.1. Основные функции УЗ

Программа УПРАВЛЕНИЕ ЗАДАНИЯМИ (УЗ) постоянно хранится в библиотеке абсолютных модулей (CL) и вызывается оттуда СУПЕРВИЗОРОМ в следующих случаях:

— после завершения *первоначальной загрузки* ДОС,

— после нормального завершения шага задания или задания в целом,

— после обработки СУПЕРВИЗОРОМ прерывания от ненормально завершенной программы (т. е. когда по той или иной причине выполнение программы было прервано до ее нормального завершения),

— по требованию оператора начать пакетную обработку в разделах переднего плана.

СУПЕРВИЗОР помещает УЗ точно на то место, где будет работать программа, предназначенная для выполнения в очередном шаге задания. Для размещения УЗ требуется 10К байтов памяти. Если невозможно реализовать разделы переднего плана таких больших размеров, создают один раздел вместо двух, ограничиваются одним фоновым разделом или организуют в разделах переднего плана выполнение одиночных программ вместо пакетированных заданий.

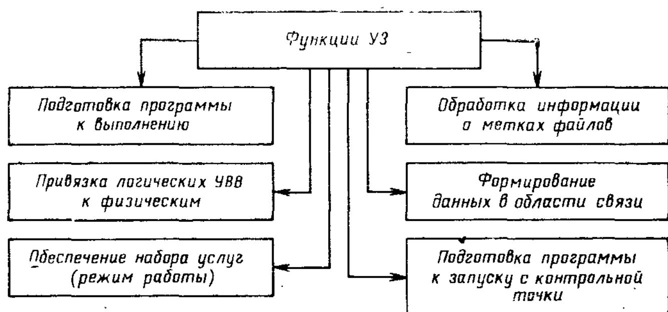


Рис. 2.1. Функции программы УПРАВЛЕНИЕ ЗАДАНИЯМИ.

Так как программа УЗ должна работать в любом из трех разделов памяти, то она построена как самоперемещающаяся.

Основные функции УЗ (рис. 2.1) обеспечивают выполнение ее назначения — управления потоком заданий.

Выполнение этих функций основывается на информации, которую УПРАВЛЕНИЕ ЗАДАНИЯМИ получает от программиста в виде *управляющих операторов*, а от оператора машины в виде *директив*.

Ниже мы рассмотрим только основные функции программы УПРАВЛЕНИЕ ЗАДАНИЯМИ. Сведения о запуске программы с контрольной точки, а также формирование области связи здесь излагаться не будут.

2.2. Подготовка программы к выполнению

Подготовка программы к выполнению проводится УЗ после восприятия оператора, указывающего наименование первой фазы затребованной программы (программа может состоять и из единственной фазы). Все

готовые к выполнению фазы ДОС/ЕС хранит в библиотеке абсолютных модулей (CL), в резидентном пакете дисков. При подготовке программы к выполнению УЗ просматривает оглавление CL и делает из него выборку записей, касающихся всех фаз выполняемой программы. Эта выборка — *оглавление фаз* посылается на резидентный диск и используется в дальнейшем СУПЕРВИЗОРОМ.

Далее УЗ очищает всю память раздела, кроме первых 150 байтов, и передает управление СУПЕРВИЗОРУ для вызова первой фазы. Все последующие фазы выполняемой программы СУПЕРВИЗОР вызывает самостоятельно без участия УЗ, используя оглавление фаз и операторы вызова, содержащиеся в обрабатываемой программе.

При подготовке программы к выполнению УЗ использует оператор начала задания

// JOB <имя задания> <комментарии>

и оператор шага задания

// EXEC [<имя программы>]

Имя в обоих случаях — идентификатор, состоящий из 1—8 символов. Имя программы в операторе EXEC не указывается, если выполняемая программа только что подготовлена РЕДАКТОРОМ (см. ниже описание режима LINK) и должна немедленно выполняться.

Символы // помещаются в первой и второй колонках перфокарты. Между ними и кодом оператора (в нашем случае — кодом JOB или EXEC) оставляется один или несколько пробелов. Пробелом или несколькими пробелами отделяется также код оператора от поля операндов, а операнды, если они требуются, отделяются друг от друга запятыми. Появление в поле операндов пробела рассматривается как признак конца этого поля, и дальнейший текст игнорируется. В некоторых случаях поле можно использовать для комментариев.

2.3. Операторы назначения и переназначения УВВ

Назначения логическим устройствам физических, как уже указывалось в § 1.4, имеют три приоритетных уровня: стандартные, постоянные и временные назначения. Программист с помощью операторов УЗ задает только

временные назначения. После окончания работы над его заданием временные назначения теряют силу и устройствам возвращаются те назначения, которые действовали до вмешательства программиста.

Кроме упоминавшихся выше символических имен логических УВВ, в директивах и операторах УЗ иногда употребляются имена SYSIN и SYSOUT. Это допустимо только в операторах и директивах ASSGN, CLOSE и EXTENT, если для SYSRDR и SYSIPT или для SYSLST и SYSPCH назначено одно и то же физическое УВВ.

Назначение для SYSIN магнитной ленты может быть как постоянным, так и временным. При назначении же диска (см. § 1.4) оно обязательно должно быть постоянным. Программист, если ему нужно такое назначение, использует оператор STOP и в примечании просит оператора выполнить с пульта постоянное назначение или вносит во вводной поток вместо оператора директиву (без символов //).

Для SYSOUT допускается назначение только магнитной ленты, и это назначение должно быть всегда постоянным. Также только постоянными должны быть назначения участков диска для системных логических устройств SYSRDR, SYSIPT, SYSLST и SYSPCH.

Назначения для логических устройств производятся директивой или оператором ASSGN, имеющим формат

$$[//]\ \text{ASSGN SYSxxx,} \left\{\begin{array}{c}\text{X'cuu'}\\ \text{UA}\\ \text{IGN}\end{array}\right\}[\text{,ALT}][\text{,TEMP}]$$

Символами SYSxxx обозначается логическое, а символами X'cuu' — назначаемое ему физическое устройство. Фигурные скобки указывают, что из перечисленных возможностей надо выбрать одну. При выборе IGN обращение к этому логическому устройству игнорируется, а при выборе UA — обращение к нему вызывает отмену задания в целом.

Операнд ALT применяется при работе с многотомными файлами (см. ниже), когда одному ЛУВВ назначаются два или более ФУВВ, на которых устанавливаются тома с продолжениями файла.

Операнд TEMP введен для того, чтобы не только программист, но и оператор мог сделать временные назначения (при использовании ASSGN как директивы оператора, вводимой с SYSLOG, символы // не набира-

ются). Оператор или программист может отменить все вре́менные назначения или часть из них, восстановив постоянные или стандартные назначения директивой (оператором):

$$[//]\ \text{RESET} \left\{ \begin{array}{c} \text{SYS} \\ \text{PROG} \\ \text{ALL} \\ \text{SYSxxx} \end{array} \right\}$$

Операнд SYS распространяет действие директивы (оператора) RESET на системные УВВ, PROG — на устройства программиста, ALL — на все УВВ машины, а а SYSxxx (здесь xxx — три конкретных символа) ограничивает это действие конкретно указанным устройством.

ДОС позволяет оператору запретить использование того или иного физического УВВ директивой

DVCDN X'cuu'

и отменить запрет директивой

DVCUP X'cuu'

позволяет немедленно отменить использование логических УВВ в одном или обоих разделах переднего плана

UNA {F1[,F2]|F2[,F1]}

а также выдать на печать действующие в данный момент назначения

LISTIO {SYS | PROG | F1 | F2 | ALL | SYSxxx | UNITS | DOWN | UA | X'cuu'}

Это действие может выполнить и программист, использовав *оператор* LISTIO (с двумя косыми чертами и пробелом впереди). Смысл операндов SYS, PROG, ALL, SYSxxx здесь тот же, что и в директиве (операторе) RESET — они обеспечивают печать списка назначений определяемых ими логических УВВ.

Операнды F1, F2 обеспечивают печать таблицы назначений УВВ для соответствующих разделов переднего плана. Операнд UNITS позволяет напечатать таблицу назначений всех ФУВВ, операнд DOWN — таблицу ФУВВ, использование которых запрещено, а UA — таблицу ФУВВ, не имеющих назначений.

Операнд X'cuu' обеспечивает печать назначений для таблицы конкретно указанного ФУВВ.

Оператор (директивой CLOSE) и программист (оператором // CLOSE) может закрыть используемые им файлы, т. е. сделать их недоступными для дальнейшего использования. Одновременно можно выполнить некоторые назначения и переназначения.

Формат директивы (оператора) CLOSE

[//] CLOSE SYSxxx[,{X'cuu'|UA|IGN|ALT}]

причем значения операндов в ней те же, что и в описанных выше операторах.

Пример 4. Программист в процессе составления задания хочет для SYSLST назначить магнитную ленту, которую в дальнейшем собирается использовать в другом ВЦ. При продолжении работы в качестве SYSLST ему необходимо АЦПУ. Написать операторы, необходимые для указанной работы.

Решение.

```
// JOB ПРИМ4
// ASSGN SYSLST,X'282'
. . . . . . . . . . . . .
// CLOSE SYSLST,UA
// RESET SYSLST
. . . . . . . . . . . . .
/&
```

С помощью оператора ASSGN производится временное назначение для системной печати — информация, предназначенная для печати, переписывается на магнитную ленту. Оператор CLOSE обеспечивает закрытие созданного на ленте файла, в данном случае перемотку ленты на начало и запрет пользования этим устройством. Оператор RESET восстанавливает действовавшее ранее назначение для SYSLST, даже если оно и не было известно программисту.

2.4. Установка режимов работы системы

Режим — это совокупность услуг, которые ДОС может предоставить программисту или оператору. Как и назначение физических УВВ логическим, режимы могут быть стандартными, постоянными и временными в зависимости от этапа и способа их введения. Режимы вводятся оператором

// OPTION <режим>[,<режим>]...

В ДОС/ЕС имеются следующие режимы, вводимые с помощью программы УЗ:

1. Режим LOG обеспечивает печать на пультовой пишущей машинке SYSLOG операторов и директив УЗ и ИОП, вводимых с перфокарт. Вводится этот режим *ди-*

рективой LOG, а отменяется директивой NOLOG. Его не следует смешивать с режимом // LOG, описываемым ниже.

2. Режим // LOG обеспечивает печать на SYSLST как управляющих операторов программиста, так и директив оператора, вводимых с SYSLOG, а также с перфокарт (между // и LOG не менее одного пробела). Этот режим может быть введен на этапе генерации ДОС как стандартный. Программист может его ввести или отменить на время выполнения своего задания с помощью операторов

```
// LOG
// OPTION LOG
// NOLOG
// OPTION NOLOG
```

Работа ДОС в режиме // LOG позволяет программисту восстановить поведение системы при выполнении его задания.

3. Режим DUMP при ненормальном завершении задания или шага задания выдает на SYSLST содержание закрепленного за ним раздела основной памяти. Как и предыдущий, режим DUMP может быть сделан стандартным. Программист его вводит и отменяет с помощью операторов

```
// OPTION DUMP
//OPTION NODUMP
```

Выдача на печать содержимого памяти может, конечно, помочь программисту в отыскании его ошибок, но этот способ следует считать крайней мерой из-за большого объема выдаваемой информации и трудности ее восприятия человеком.

4. Режим LINK обеспечивает редактирование программы и ее выполнение без запоминания в библиотеке абсолютных модулей. Отдельные модули, поступающие после трансляции на редактирование, записываются на SYSLNK, откуда они и считываются программой РЕДАКТОР. Режим этот пригоден только для программ, состоящих из одной фазы. Стандартным режим LINK не бывает, а поэтому он должен всегда вводиться программистом с помощью оператора

```
// OPTION LINK
```

а отменяться оператором

// OPTION NOLINK

5. Режим CATAL, как и LINK, не бывает стандартным и устанавливается для возможности запоминания в CL вырабатываемых РЕДАКТОРОМ фаз программы. Создаваемые при использовании этого режима программы могут после этого быть вызваны для выполнения согласно присвоенным им именам.

Вводится этот режим оператором

// OPTION CATAL

а отменяется оператором

// OPTION NOLINK

т. е. так же, как отменяется режим LINK.

Существует еще несколько реже употребляемых режимов работы системы, перечень которых можно найти в справочнике[1].

Кроме режимов работы ДОС в целом программа УЗ позволяет установить режимы работы трансляторов. Все они вводятся оператором // OPTION наравне с режимами работы ДОС.

Режим работы трансляторов можно определить правилом

<режим> : : =DECK | LIST | SYM | XREF | ERRS | LISTX | NODECK | NOLIST | NOSYM | NOXREF | NOERRS | NOLISTX | 48C | 60C

Все эти режимы могут быть установлены как стандартные при генерации системы.

Услуги, обеспечиваемые каждым из этих режимов (кроме режимов, их отрицающих, например NOLIST, NODECK и т. д.), описаны в табл. 2.1.

Пример 5. Требуется транслировать несколько программ:

— первую из них выдать после трансляции на перфокарты;

— вторую каталогизировать в библиотеку абсолютных модулей для дальнейшего использования;

— третью и четвертую объединить в единое целое и использовать сразу после сформирования.

---

[1] Операционная система ДОС/ЕС. Справочник. М., «Статистика», 1977.

Таблица 2.1

Режим	Услуга	Транслятор	УВВ
DECK	Вывод объектных модулей на перфокарты, диски и пр.	Все языки	SYSPCH
LIST	Печать исходных модулей	»	SYSLST
LISTX	Печать объектных модулей в „ассемблироваином" виде	ПЛ/1, ФОРТРАН IV	SYSLST
SYM	Вывод таблицы символов	АССЕМБЛЕР, ПЛ/1	SYSPCH SYSLST
XREF	Вывод таблицы перекрестных ссылок	То же	»
ERRS	Выдача списка ошибок в программе	Все языки	»
48C	Определение алфавита, используемого для кодироваиия исходного модуля ПЛ/1	ПЛ/1	—
60C		ПЛ/1	—

Решение. Для первой программы необходим режим DECK, который не нужен для других программ. Режим CATAL, необходимый для второй программы, должен тоже быть отменен до начала работы над объединением третьей и четвертой программы. Поэтому следует подготовить такой набор управляющих перфокарт:

```
// JOB ПРИМ5
// OPTION DECK
// EXEC <имя транслятора>
     <перфокарты с текстом первой программы>
/*
* СИМВОЛЫ /* ОБОЗНАЧАЮТ ОКОНЧАНИЕ ТЕКСТА ПРО-
* ГРАММЫ
// OPTION NODECK, CATAL
* МЫ ОДНОВРЕМЕННО ОТМЕНЯЕМ РЕЖИМ DECK И ВВО-
* ДИМ РЕЖИМ CATAL
// EXEC <имя транслятора>
* ДО ОПЕРАТОРА EXEC МЫ ДОЛЖНЫ БЫЛИ ИСПОЛЬЗО-
* ВАТЬ ОДИН ИЗ ОПЕРАТОРОВ РЕДАКТОРА, ОПРЕДЕЛЯЮ-
* ЩИЙ ИМЯ СОЗДАВАЕМОЙ ПРОГРАММЫ, НО ЭТОТ ВОП-
* РОС МЫ ОТЛОЖИМ ДО ГЛАВЫ 5
     <перфокарты с текстом второй программы>
/*
// OPTION NOLINK
// OPTION LINK
* ПЕРВЫМ ОПЕРАТОРОМ OPTION МЫ ОТМЕНЯЕМ НЕ РЕ-
* ЖИМ LINK, А РЕЖИМ CATAL, А ВТОРЫМ ВВОДИМ РЕЖИМ
* LINK
// EXEC <имя траислятора>
     <перфокарты с текстом третьей программы>
/*
// EXEC <имя транслятора>
     <перфокарты с текстом четвертой программы>
```

```
/*
// EXEC <имя программы-РЕДАКТОРА>
// EXEC
    <перфокарты с исходными данными, если они требуются>
/*
/&
```

2.5. Редактирование и запоминание информации о метках файлов

Как уже упоминалось в § 1.2, данные в ДОС/ЕС представляются в виде записей, блоков, файлов и томов. Наименьшей порцией данных, которой может управлять логическая система управления вводом — выводом (ЛСУВВ), является запись. Работать с отдельными элементами записи в ДОС/ЕС могут только программы пользователя. Физическая же система управления вводом — выводом (ФСУВВ) работает с томами и блоками.

Файлы и тома данных на дисках и лентах снабжаются метками, т. е. специальными опознавательными записями, облегчающими их использование.

На магнитных лентах различают *метки тома* и *метки файла*, *головные* и *хвостовые*, *стандартные* и *нестандартные*, *системные метки* и *метки*, *принадлежащие пользователю*. Метки файлов на ленте размещаются непосредственно до и после файлов. На пакетах дисков все метки сосредоточены в первом записанном на диске файле — *оглавлении тома*.

Задача ДОС — при создании нового файла снабдить его метками, а при использовании уже созданного файла обеспечить чтение меток и принятие решений на основе их анализа.

Снабжение метками создаваемого ДОС файла производится на основе информации, передаваемой программистом системе через управляющие операторы

// TLBL <операнды>

для файлов на лентах и

// DLBL <операнды>
// EXTENT <операнды>
для файлов на дисках.

Среди набора операндов оператора TLBL обязательно должен присутствовать только первый, задающий имя файла. Смысл каждого из операндов приводится ниже:

<операнды> : : =<имя файла>,<'идентификатор

файла'>,<дата>,<регистрационный номер файла>, <порядковый номер тома>,<порядковый номер файла>,<номер поколения>,<номер версии>

Все операнды оператора TLBL — позиционные, т. е. они распознаются по их месту в списке. Пропускаемый операнд отмечается запятой. Но пропуск некоторого числа последних операндов позволяет пропускать и запятые.

Операнд <имя файла> должен состоять из 1—8 символов (среди которых не должно быть пробелов) и совпадать с его именем в макрокоманде DTF, описывающей этот файл, или с его именем в описании на алгоритмическом языке.

Операнд <'идентификатор файла'> дополняет возможности опознавания файла и присваивается ему пользователем при создании. Этот операнд может состоять из 1—17 символов, включая возможные пробелы, и должен ограничиваться апострофами.

Операнд <дата> обозначает количество дней или абсолютный срок хранения файла. При попытке произвести запись в файл, срок хранения которого не истек, ДОС выдает сообщение об этом на SYSLOG и требует от оператора указаний. Операнд <дата> записывается в виде 1—4 десятичных цифр для количества дней хранения и гг/ддд — для абсолютного срока хранения файла. Здесь гг — последние две цифры года, а ддд — порядковый номер дня в году. Если дата не указана, то файл хранится 7 дней. Операнд <регистрационный номер файла> состоит из 1—6 символов (не обязательно цифр), он должен совпадать с номером тома, в котором находится файл, а для многотомных файлов — с номером тома, где хранится начало файла.

Таким образом, номер тома, как и регистрационный номер файла, не обязательно должен быть числом (его не следует смешивать с операндом <порядковый номер тома>, рассматриваемым ниже).

Операнд <порядковый номер тома> должен содержать от 1 до 4 десятичных цифр, причем нумерация томов, в которых расположен файл, начинается с 0001.

Аналогично написание операнда <порядковый номер файла>.

Последние два операнда оператора TLBL позволяют создавать несколько вариантов файла с одним и тем же наименованием, например, несколько вариантов находя-

щейся в процессе отладки программы, различных по времени составления вариантов списка личного состава предприятия и т. п. Более существенные изменения отмечаются изменением номера поколения, который изображается 1—4 цифрами, а более мелкие — изменением номера версии (1—2 цифрами).

Пример 6. Определить файл TABLE на магнитной ленте 123456, где уже записано 10 файлов, указав его принадлежность П. П. Петрову, срок хранения 6 мес. Версию и поколение не указывать.

Решение.

// TLBL TABLE,'П.П.Петров',180,123456,1,11

Пример 7. Определить файл «VEKTOR» на ленте, где еще нет ни одного файла. Указать, что поколение и версия нулевые.

Решение.

// TLBL VEKTOR,,,,1,1,0,0

Для оператора DLBL

<операнды> : : =<имя файла>,<'идентификатор файла'>,<дата>,<коды>,<признак секретности файла>

причем обязательным является только первый из операндов.

Операнд <имя файла>, как и в операторе TLBL, содержит от 1 до 8 символов без пробелов, а операнд <'идентификатор файла'> всегда ограничивается апострофами, он может содержать пробелы и другие символы набора ДОС/ЕС в количестве до 44. Имя файла в операторе DLBL должно совпадать с именем, присвоенным ему в макрокоманде DTF АССЕМБЛЕРА, операторе DECLARE языка ПЛ/1 и в аналогичных операторах других алгоритмических языков. Операнд <дата> аналогичен соответствующему операнду оператора TLBL. Аналогично файлам на лентах отсутствие оператора <дата> означает возможность разрушения его через 7 дней.

Операнд <коды>, где

<коды> : : ={SD|DA|ISC|ISE}

определяет тип организации файла:

SD — последовательная организация,

DA — прямая организация,

ISC, ISE — индексно-последовательная организация.

Код ISC употребляется при создании или расширении файла. При обращении к файлу с целью дополнения или замены записей следует употреблять код ISE.

Для файлов с последовательной организацией операнд <коды> можно опускать.

Для файлов с ограниченным доступом в поле операнда <признак секретности файла> необходимо применять код DSF.

Оператор DLBL содержит только часть описателей определяемого файла. Остальные описатели задаются отдельно для каждого участка файла, если он хранится в разных местах пакета дисков или на разных пакетах. Для этого используется упомянутый выше оператор EXTENT формата

// EXTENT <операнды>

для которого все операнды необязательны.

Для этого оператора

<операнды> : : =<логическое УВВ>,<регистрационный номер файла>,<тип участка>,<номер участка>,<адрес участка>,<количество дорожек>,<дорожка разделения цилиндра>

Операнд <логическое УВВ> определяет устройство, на котором установлен дисковый пакет, содержащий описываемый участок файла. Он может не указываться, если имя ЛУВВ совпадает с именем устройства для предыдущего участка или если оно было определено в макрокоманде DTF АССЕМБЛЕРА (оператора DECLARE ПЛ/1, оператора DEFINE FILE ФОРТРАНа и пр. (см. гл. 4)).

Операнд <регистрационный номер файла> содержит имя пакета дисков (1—6 символов), на котором записан участок файла.

Операнд <тип участка> может принимать значения 1, 2, 4 и 8. Тип 1 присваивается участку, предназначенному для размещения данных, тип 2 отводится для области переполнения индексно-последовательного файла, тип 4 — для размещения индексной области и, наконец, тип 8 — для размещения данных при разделении цилиндров между двумя и более файлами. Такой способ размещения файлов (рис. 2.2) удобен, если соответствующие элементы данных нескольких файлов должны извлекаться с диска или посылаться на него одновременно.

Операнд <номер участка> может принимать значения в пределах от 0 до 255 включительно, а операнд <адрес участка> изображается 1—5 десятичными цифрами. Адресом участка называется номер первой при-

надлежащей файлу дорожки при их сквозной нумерации (цилиндр за цилиндром) в порядке возрастания их номеров. Начинается нумерация как цилиндров, так и дорожек с нуля, нулевая дорожка, например, в 19-м цилиндре это 190-я, 5-я дорожка в 73-м цилиндре — 735-я и т. д.

Операнд <количество дорожек> определяет количество дорожек во всех входящих в участок цилиндрах. Для участков типа 1 — это число цилиндров, умноженное на 10, для участков типа 8 оно определяется разделением цилиндров.

Операнд <дорожка разделения цилиндров> содержит номер последней дорожки, выделенной описываемому в операторе EXTENT участку файла (у изображенных на рис. 2.2 файлов дорожки разделения цилиндров соответственно имеют значения 2, 4 и 9).

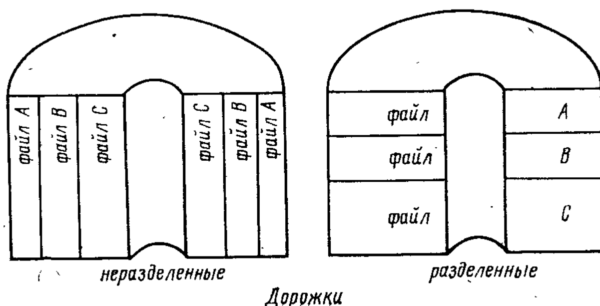


Рис. 2.2. Размещение файлов на дисках. Справа разделенные цилиндры

Пример 8. Разместить на пакете дисков в цилиндрах 30—41 два файла с прямой организацией под именами ABC12 и ABC13. Идентификаторов не требуется. Файлы должны храниться два года. Регистрационный номер пакета — ИВАНОВ1. Имя ЛУВВ — SYS002. Объем файлов соответственно 5 и 7 цилиндров.

Решение.

```
// DLBL ABC12,,730,DA
// EXTENT SYS002,ИВАНОВ1,1,0,300,50
// DLBL ABC13,,730,DA
// EXTENT SYS002,ИВАНОВ1,1,0,350,70
```

Пример 9. Два файла с последовательной организацией требуется разместить так, чтобы ими можно было пользоваться в одних и тех же местах программы. Имена файлов ABC14 и ABC15. Доступ к файлам ограничен. Владелец файлов В. А. Петров. Размещать файлы можно на пакете дисков предыдущего примера с 42 по 62 цилиндры.

Решение.

```
// DLBL ABC14,'В.А.ПЕТРОВ',,,DSF
// EXTENT SYS002,ИВАНОВ1,1,8,420,105,4
// DLBL ABC15',В.А.ПЕТРОВ',,,DSF
// EXTENT SYS002,ИВАНОВ1,1,8,425,105,9
```

При размещении файлов предполагалось, что они занимают равный или примерно равный объем. Поэтому для каждого из них выделено по 5 дорожек в цилиндре: с 0-й по 4-ю для первого и с 5-й по 9-ю для второго. Фамилия их владельца использована в качестве идентификатора. Ограниченность доступа отражена наличием в операторе DLBL операнда секретности.

Пример 10. Выделить в пакете ИВАНОВ1 место для размещения файла с индексно-последовательной организацией: 1 цилиндр для области индекса и 20 — для области данных. Файл хранить бессрочно.

Решение.

```
// DLBL ABC16,,99/365,ISC
// EXTENT SYS002,'ИВАНОВ1,'4,1,630,10
// EXTENT SYS002,'ИВАНОВ1',1,2,640,200
```

Файлы с индексно-последовательной организацией или короче индексно-последовательные файлы (подробно описанные в гл. 7) состоят из области индекса и области данных. Области другого назначения могут в таких файлах отсутствовать. Размещать эти области на диске надлежит подряд и в таком порядке, как это показано в приведенном решении.

Операнд <тип участка> для области индекса должен быть равен 4, а для области данных — 1.

Для запоминания информации о файлах, используемых в каком-либо из разделов, а также о файлах общего пользования на резидентном пакете дисков выделен цилиндр меток — 10 дорожек, каждой из которых отведена особая роль. Четные дорожки 0, 2, 4 предназначены для хранения временной информации, т. е. информации о файлах, которые предполагается использовать лишь в текущем задании. Нечетные же дорожки 1, 3, 5, наоборот, служат для записи информации о файлах, которые могут потребоваться в ряде заданий. Дорожки с номерами 6—9 отведены для системных файлов.

Использовав управляющий оператор // OPTION с операндом URSLABEL мы указываем ДОС, что информация о метках временная. Для постоянной информации применяется операнд PARSTD. Наконец, чтобы информацию о метках записать в системную область цилиндра меток, используется операнд STDLABEL.

Каждому разделу в цилиндре меток выделены одна дорожка для хранения временной информации и одна для хранения постоянной. Описанное распределение дорожек приводится в табл. 2.2.

Таблица 2.2

Тип информации о метках файлов	Вид операнда в операторе//OPTION	Номера дорожек для			
		BG	F2	F1	системных файлов
Временная	URSLABEL	0	2	4	—
Постоянная	PARSTD	1	3	5	—
Принадлежащая системным файлам	STDLABEL	—	—	—	6—9

При отсутствии у оператора // OPTION операндов, определяющих порядок использования файлов, все определяемые файлы считаются временными и информация о них в зависимости от раздела заносится на дорожки 0, 2 и 4. Информация, записанная на четной дорожке, может использоваться в любом шаге текущего задания, но с окончанием задания разрушается. На каждом шаге задания имеется возможность записывать временную информацию о новых файлах на место записанной на предыдущих шагах, разрушая прежнюю запись.

Постоянная информация после окончания задания не разрушается и может использоваться в последующих заданиях, но последующие шаги текущего и очередных заданий, записывая свою постоянную информацию, разрушают прежнюю.

Аналогично хотя информация о системных файлах делает их доступными всем разделам на протяжении ряда заданий, но и она не защищена от разрушения записью новой информации о системных файлах.

Используется информация о метках следующим образом:

— по имени файла запись на цилиндре меток ищется в области временных записей своего раздела;

— если она найдена, поиск прекращается, в противном случае поиск продолжается сначала в области постоянных, а затем в области системных записей.

Пример 11. Все файлы данного задания передать другому заданию без повторного написания операторов // DLBL и // EXTENT. Ввести режим выдачи таблицы символических имен транслируемых программ.

Решение.

```
// JOB ПРИМ11
// OPTION PARSTD
// OPTION SYM
. . . . . . . . . . . . . .
/&
```

2.6. Составление комплексных заданий

Пример 12. Необходимо транслировать программу, записанную на магнитной ленте X'280' в виде файла без меток. Результат трансляции выдать на перфокарты. Для SYSRDR и SYSIPT имеется постоянное назначение X'00C', для SYSPCH — X'00E'.

Решение.

```
// JOB ПРИМ12
// ASSGN SYSIPT,X'280'
// OPTION DECK
// EXEC <имя транслятора>
// RESET SYSIPT
/&
```

Назначив X'280' для SYSIPT, мы потом восстанавливаем ранее действовавшее постояное назначение. После оператора

// EXEC <имя транслятора>

перфокарты не подложены, так как они поступают с устройства X'280'.

Пример 13. Имеется некоторая программа АБВГДЕЖЗ. Ее исходные данные хранятся в пятом файле тома 123456 на магнитной ленте, который программа ищет на SYS007 под именем TP112. Вывод результатов в виде файла TP113 программа осуществляет на SYS008, под которым понимается пакет дисков № 654321, цилиндры с 20-го по 25-й. Результат хранить 1 мес. Файл передать следующему заданию пакета.

Решение.

```
// JOB ПРИМ13
// TLBL TP112,,,123456,,5
// OPTION PARSTD
// DLBL TP113,,30
// EXTENT SYS008,654321,1,0,200,60
// ASSGN SYS007,X'282'
// ASSGN SYS008,X'192'
// EXEC АБВГДЕЖЗ
/&
```

Пример 14. Транслировать 6 программ:

— первые две написаны на ФОРТРАНЕ, третья на языке АССЕМБЛЕРА, а остальные — на ПЛ/1;

— программы поступают: первая с магнитной ленты предыдущего примера из пятого файла, следующие четыре через системный ввод, а последняя с пакета дисков. Имена программ ABC11, ABC12, ABC13, ABC14, ABC15, ABC16;

— результаты трансляции выдать на перфокарты и отпечатать.

Решение.

```
// JOB ПРИМ14
// OPTION DECK, LIST
// TLBL ABC11,,,123456,,5
// ASSGN SYSIPT,X'282'
// MTC REW,X'282'
// MTC FSF,X'282',5
* ОПЕРАТОР MTC, ПОДРОБНО ОПИСЫВАЕМЫЙ В ГЛ. 7, ПО-
* ЗВОЛЯЕТ УПРАВЛЯТЬ ДВИЖЕНИЕМ МАГНИТНОЙ ЛЕНТЫ
* НА ФИЗИЧЕСКОМ УРОВНЕ. В ДАННОМ СЛУЧАЕ ЛЕНТА
* БЫЛА ПЕРЕМОТАНА НА НАЧАЛО, А ПОТОМ ПОД СЧИТЫ-
* ВАЮЩИЕ ГОЛОВКИ БЫЛ ПОДВЕДЕН ФАЙЛ С НОМЕ-
* РОМ 5.
// EXEC FORTRAN
// RESET SYSIPT
// EXES FORTRAN
   <перфокарты программы ABC12>
/*
// EXEC ASSEMBLY
   <перфокарты программы ABC13>
/*
// EXEC PL/I
   <перфокарты программы ABC14>
/*
// EXEC PL/I
   <перфокарты программы ABC15>
/*
// DLBL ABC16
// EXTENT SYSIPT,654321,1,0,300,20
   ASSGN SYSIPT,X'192'
// EXEC PL/I
   ASSGN SYSIPT,X'00C'
```

Назначение для SYSIPT перед трансляцией с ПЛ/1 производится не оператором // ASSGN, а директивой, которая хотя и расположена среди операторов программиста, но не имеет двух косых черточек «//», что заставляет программу УПРАВЛЕНИЕ ЗАДАНИЯМИ делать постоянные назначения. А как сказано в § 1.4, для системных логических УВВ (SYSRDR, SYSIPT, SYSLST, SYSPCH) временные назначения на дисковые устройства не разрешены. Но, сделав при выполнении текущего задания какое-либо постоянное назначение, программист обязан таким же способом его и отменить, чтобы не нарушить прохождение последующих заданий.

Пример 15. Построить программу из двух исходных модулей, каждый из которых написан на ФОРТРАНе. Текст программы не нужен, но печать возможных ошибок желательна. Исходные модули поступают с магнитной ленты № 222222, где они являются файлами № 12 и 13. Исходные данные программы — на перфокартах. Программу выполнять сразу после ее сформирования.

Решение.

```
// JOB ПРИМ15
// OPTION NOLIST,ERRS
// TLBL TP112,,,222222,,12
// ASSGN SYSIPT,X'283'
// MTC REW,X'283'
// MTC FSF,X'283',12
// OPTION LINK
// EXEC FORTRAN
// TLBL TR112,,,222222,,13
// EXEC FORTRAN
// DLBL PCHABC,,99/365
// EXTENT SYSPCH,333333,1,0,440,10
   ASSGN SYSPCH,X'190'
// EXEC <имя первой фазы программы РЕДАКТОР>
// RESFT SYSIPT
// EXEC
   <исходные данные программы>
/*
   ASSGN SYSPCH,X'00C'
/&
```

2.7. Порядок следования операторов программы УЗ

Примеры, приведенные в § 2.5, хорошо иллюстрируют порядок следования операторов:

— каждое задание начинается с оператора

// JOB <имя задания>

и заканчивается оператором

/&

— каждый шаг задания заканчивается оператором

// EXEC [<имя программы>]

а перфокарты с исходными данными и оператор конца файла —

/*

должна находиться не в SYSRDR, а в SYSIPT;

— все остальные операторы определенного шага располагаются перед своим оператором // EXEC в произвольном порядке, причем операторы

// DLBL <операнды>

должны предшествовать своим операторам

// EXTENT <операнды>

Операторы // TLBL и // DLBL могут располагаться как перед, так и после операторов (или директив) // ASSGN, которыми делаются назначения соответствующих устройств.

3. Языки программирования ДОС/ЕС

3.1. Общие понятия

Одна из первых версий ДОС/ЕС в качестве средств автоматизации программирования имела программу АССЕМБЛЕР и трансляторы с трех языков: РПГ, Базисного ФОРТРАНа и подмножества ПЛ/1. В дальнейших версиях были добавлены трансляторы с языков ФОРТРАН-4 и КОБОЛ.

Известно, что язык ФОРТРАН, а также АЛГОЛ-60 были созданы для написания алгоритмов и программ решения так называемых научно-технических задач, для которых характерен большой объем вычислительных операций и сравнительно малый объем операций ввода—вывода и пересылки данных. Для решения же коммерческих задач, наоборот, требуется большой объем работ по вводу, сортировке, выборке и выводу данных при сравнительно малом удельном весе вычислительных операций. Для решения таких задач в 1961 г. был создан язык КОБОЛ.

Но продолжающееся расширение возможностей ЭВМ и эволюция как научно-технических, так и коммерческих задач привели к необходимости создания языка, в равной степени пригодного и для вычислений, и для сортировочно-пересылочных операций. Одним из языков этого типа является ПЛ/1. Кроме возможностей АЛГОЛа-60, ФОРТРАНа и КОБОЛа, он располагает средствами обработки списков и некоторыми другими возможностями.

Программисты, работающие в вычислительной среде ДОС/ЕС, т. е. готовящие программы и решающие задачи на тех ЭВМ, управление которыми осуществляется посредством ДОС/ЕС, должны знать не только языки программирования в чистом виде, но и конкретные способы их использования в данных условиях.

ДОС/ЕС, как и ОС/ЕС, базируется на языке АССЕМБЛЕРА, а наиболее высоким по уровню алгоритмическим языком, использование которого возможно в рамках этих систем, является язык ПЛ/1 (в частности, в ДОС/ЕС — подмножество ПЛ/1).

Общеизвестно, что хотя язык АССЕМБЛЕРА и позволяет наиболее полно использовать возможности машины, но программирование на нем по трудоемкости сравнимо с программированием в машинных кодах. Пре-

имуществом использования АССЕМБЛЕРА является возможность поручения АССЕМБЛЕРУ распределения памяти, для чего вместо конкретных адресов используются символические выражения. Что же касается мнемоники языка АССЕМБЛЕРА в части кодов операций, то она рассчитана на программиста, знающего английский язык; это создает неудобства для программистов, не знающих этого языка. Для такого программиста идеалом, пожалуй, являются языки АЛГОЛ-60 и ФОРТРАН, в лексиконе которых имеется не более трех десятков английских слов (их можно рассматривать как специальные обозначения). В лексиконе же ПЛ/1 насчитывается около 200 таких слов, что объясняется его чрезвычайно широкими по сравнению с АЛГОЛом-60 и ФОРТРАНом возможностями.

Вместе с тем структура этих трех языков имеет много общего и совместное их изучение позволяет получить значительный выигрыш как во времени, так и в полноте усвоения. Кроме того, практика показывает, что для современного программиста знания одного языка, будь то ФОРТРАН, АЛГОЛ, ПЛ/1 или какой-либо другой язык, недостаточно.

В настоящей главе ведется совместное изложение элементов и средств языков АЛГОЛа-60, Базисного ФОРТРАНа и подмножества ПЛ/1. Такой выбор имеет следующее обоснование:

— изложение алгоритмических языков ДОС/ЕС совместно с АЛГОЛом (который к ДОС/ЕС не привязан) облегчает их усвоение, так как АЛГОЛ знает большинство программистов. Для тех же, кто его не знает, это облегчит его использование при переходе от ДОС/ЕС к ОС/ЕС;

— языки Базисный ФОРТРАН и подмножество ПЛ/1 включены в первую версию ДОС/ЕС и трансляторы с них входят в состав программного обеспечения всех машин младших моделей ЕС ЭВМ;

— имеющийся в той же версии ДОС/ЕС язык РПГ сильно отличается от любого из языков, созданных на базе триады АЛГОЛ — ФОРТРАН — КОБОЛ, и не имеет особых перспектив распространения среди программистов;

— переход от Базисного ФОРТРАНа к ФОРТРАНу IV, включенному в более поздние версии ДОС/ЕС, для программистов затруднений не составляет;

— отсутствие ниже описания КОБОЛа объясняется тем, что с точки зрения автора его достоинства перекрываются ПЛ/1 (даже в варианте подмножества, реализованного в ДОС/ЕС).

Ниже сначала рассматриваются структура программы в целом, потом используемые символы и способы построения из них имен и констант. Далее изучаются выражения, операторы (в АЛГОЛе — описания и операторы) и более сложные структуры: группы, блоки, составные операторы, подпрограммы (процедуры).

3.1.1. Средства описания языков. Как известно, язык описания языков называется метаязыком. Впервые это понятие применительно к естественным языкам было использовано Н. Хомским при описании языка в виде набора правил [1])

$$\begin{array}{l} <a_1> \longrightarrow <b_{11}>, \\ <a_1> \longrightarrow <b_{12}>, \\ \dots\dots\dots \\ <a_1> \longrightarrow <b_{1m}>, \\ <a_2> \longrightarrow <b_{21}>, \end{array} \qquad (3.1)$$

в которых слева стоят более общие понятия языка — *металингвистические переменные*, а справа — более частные понятия: либо также металингвистические переменные, являющиеся левыми частями других правил, либо понятия, не подлежащие разъяснению, так называемые *терминальные понятия*.

Метаязык Хомского был развит Бэкусом и Науром применительно к описанию языка АЛГОЛ-60, что явилось качественно новым уровнем строгости описания языка. В нотации Бэкуса — Наура, известной всем программистам, выражение (3.1) получает вид

$$\begin{array}{l} <a_1> ::= <b_{11}> | <b_{12}> | \dots | <b_{1m}>, \\ <a_2> ::= <b_{21}> | \dots \end{array} \qquad (3.2)$$

причем символами метаязыка являются уголковые скобки $<$ и $>$, ограничивающие металингвистические переменные; вертикальная черта |, означающая возможность выбора («или»), и знак $::=$, означающий «это есть».

В описаниях ФОРТРАНа, КОБОЛа и некоторых других языков используется иная нотация:

---

[1]) **Хомский Н.** (Chomsky N.) Three models for the description of language. — «IRE Trans. Inform. Theory», IT2, 1956. (Рус. пер. «Киберн. сб. Вып. 2, 1961).

— написание

$$\left\{\begin{matrix} a \\ b \\ c \end{matrix}\right\} \qquad (3.3)$$

означает выбор одного из элементов a, b или c; его можно также записать в виде

{a|b|c} (3.4)

— написание

[a] (3.5)

означает, что элемент a не является обязательным и может быть опущен.

В описании языка ПЛ/1 употребляется удачное сочетание обеих нотаций, дополненное многоточием..., означающим возможность повторения предшествующей конструкции один и более раз.

Характерный для ФОРТРАНа и особенно для ПЛ/1 *принцип умолчания* в описании требует подчеркивания того понятия (3.3), (3.4), которое предполагается действующим при отсутствии явного указания программиста. В АЛГОЛе-60 по умолчанию объявляются метки, а массивы без объявления типа считаются действительными.

Так, появление текста

M : X : : =0;

квалифицирует идентификатор M как метку, а текст

array X [1 : n];

равносилен тексту

real array X [1 : n];

В ФОРТРАНе и ПЛ/1 этот прием используется более широко. Например, идентификатор ABC считается имеющим описатель EXTERNAL (внешний) и STREAM (передача потоком), если он в ПЛ/1 является именем файла и если не оговорено противное (т. е. если не заданы описатели INTERNAL и RECORD). Если же XYZ обозначает имя переменной, массива или структуры, то при отсутствии описателя области действия ему неявно присваивается описатель INTERNAL (внутренний). Таким образом

⟨список описателей файла⟩ : : =

$$= \mathrm{FILE} \left\{\begin{matrix} \mathrm{RECORD} \\ \underline{\mathrm{STREAM}} \end{matrix}\right\} \left\{\begin{matrix} \underline{\mathrm{EXTERNAL}} \\ \mathrm{INTERNAL} \end{matrix}\right\}$$

3.1.2. Конструкция программы. По своему внешнему представлению в АЛГОЛе-60 и ПЛ/1 программа является строкой символов неопределенной длины. Пробелы в АЛГОЛе игнорируются, тогда как в ПЛ/1 они несут синтаксическую нагрузку, исполняя обязанности разделителей. Но количество пробелов не играет роли, и любое число пробелов эквивалентно одному.

В ФОРТРАНе программа представляет собой последовательность записей неопределенной длины, причем каждая из записей определенным образом «привязана» к позициям перфокарты.

Внутреннее представление программы в этих языках также различно. Как известно, в АЛГОЛе-60 программа представляется либо блоком, либо составным оператором, но в обоих случаях—единым целым. Такое мощное средство АЛГОЛа, как процедура должна содержаться внутри блока. В ФОРТРАНе же и ПЛ/1 программа является набором отдельных подпрограмм или процедур, одна из которых рассматривается в качестве главной.

Эти различия являются не только формальными; они не позволяют в АЛГОЛе рассматривать процедуру как законченную программу и производить сборку более крупных программ из процедур. В этом отношении ФОРТРАН, язык с меньшими возможностями, имеет преимущество перед АЛГОЛом.

Итак, в программе, написанной на АЛГОЛе-60, процедуры, если они имеются, входят в нее на правах описаний старшего или вложенных блоков. В ФОРТРАНе же и в ПЛ/1 вводится понятие *программной единицы* — самостоятельной части программы, роль которой могут играть главная программа, подпрограмма-функция и подпрограмма общего вида. При трансляции в ДОС/ЕС каждая программная единица преобразуется в отдельный объектный (или перемещаемый) модуль (см. ниже, гл. 6). В ПЛ/1 оба эти вида подпрограмм называются *процедурами* или *процедурными блоками.* Вместе с тем структура описаний процедур в АЛГОЛе-60 и структура программных единиц в ФОРТРАНе и ПЛ/1 имеют много общего, что позволяет излагать их в сравнении (см. табл. 3.1).

Отметим сразу же, что в составном операторе АЛГОЛа-60 **begin** и **end** — ключевые слова, тогда как в аналоге составного оператора — группе языка ПЛ/1

Таблица 3.1

Металингвистические переменные	Языки программирования		
	АЛГОЛ-60	Базисный ФОРТРАН	Подмножество ПЛ/1
<программа>	<блок>|<составной оператор>	<главная программа>|<главная программа>[<подпрограмма>]...	<главная процедура>|<главная процедура>[<процедура>]...
<составной оператор>	(см. табл. 3.10)	—	—
<блок>(<начальный блок>ПЛ/1)	(см. табл. 3.10)	—	(см. табл. 3.10)
<главная программа> (в ПЛ/1— <главная процедура>)	—	[<оператор>] . . .	<оператор главной процедуры>; [<предложение>];... <оператор END>;
<подпрограмма>(<процедура> в ПЛ/1)	—	(см. табл. 3.12)	(см. табл. 3.12)
<предложение>	—		(см. табл. 3.11)
<оператор главной процедуры>	—	—	<имя входа>: PROCEDURE OPTIONS (MAIN)

слова DO и END — операторы, что и подчеркивается отделением их от дальнейшего текста точкой с запятой. Аналогичное замечание должно быть сделано и относительно слова BEGIN, открывающего «обычный» (начальный) блок ПЛ/1 (см. ниже, § 3.8).

В АЛГОЛе-60 описание процедуры состоит из заголовка процедуры и тела процедуры. Тело процедуры — оператор, чаще всего блок. В описании процедуры-функции ключевому слову **procedure** предшествует описатель <тип> выдаваемого значения: **real**, **integer** или **Boolean**. Имя процедуры или процедуры-функции стоит вслед за словами **procedure.** Заголовок может содержать вслед за именем процедуры списки формальных параметров, значений и спецификаций.

Описание процедуры помещается в описательной части какого-либо блока и имеет силу только внутри этого блока.

Аналог заголовка процедуры в ФОРТРАНе является *оператором* и для процедур-функций имеет вид

[<тип>] FUNCTION <имя>[(<список формальных параметров>)]

При отсутствии описателя <тип> он определяется в соответствии с первой буквой имени функции: *целый* при I, J, K, L, M, N и *действительный* при остальных значениях первой буквы.

Для процедур общего вида аналог заголовка именуется оператором SUBROUTINE и имеет вид

SUBROUTINE <имя>[(<список формальных параметров>)]

Тело программы в обоих случаях состоит из одного или нескольких операторов, из которых хотя бы один должен быть оператором возврата к вызвавшей программе или подпрограмме, а последний — оператором конца подпрограммы.

Их вид

```
RETURN
END
```

и, как любой другой оператор ФОРТРАНа, они располагаются на отдельных перфокартах.

Формальные параметры, когда они присутствуют, являются идентификаторами и их тип описывается обычными для ФОРТРАНа способами.

В ПЛ/1 аналогом подпрограммы ФОРТРАНа является процедурный блок, заголовок которого носит название оператора PROCEDURE и имеет вид

<имя входа> : PROCEDURE[(<список формальных параметров>)] [<список описателей функции>];

Списком описателей функции исчерпывается внешнее различие между процедурой общего вида и процедурой-функцией. Различие же между ними по существу во всех трех языках состоит в том, что для АЛГОЛа и ФОРТРАНа в теле процедуры-функции ее имя должно стоять в левой части хотя бы одного оператора присваивания. Тело процедуры ПЛ/1 может иметь довольно сложный вид и для процедуры-функции должно включать операторы возврата вида

RETURN[(выражение)];

позволяющие возвращать определяемые значения в точку вызова. Заканчивается тело операторами END; или END <имя входа>;

Особенностью тела процедур ПЛ/1 (кроме главных) является возможность появления в нем *дополнительных точек входа*, оформляемых в виде оператора ENTRY:

<имя входа> : ENTRY[(<список формальных параметров>)] [<описатели функции>];

В ПЛ/1 в отличие от ФОРТРАНа процедурные блоки не обязательно должны быть отдельными программными единицами. Они могут располагаться и внутри других процедурных или обычных блоков.

3.1.3. Символика. Наборы символов, используемые рассматриваемыми тремя языками (табл. 3.2), различаются незначительно.

Базисный ФОРТРАН использует прописной латинский алфавит от A до Z, знак $, цифры от 0 до 9, круглые скобки, знак равенства, запятую, десятичную точку, знаки +, —, звездочку и косую черту. АЛГОЛ использует дополнительно строчные латинские буквы и допускает национальные алфавиты. Знак $ не используется. Для знака умножения применяется более привычный $\times$, а для возведения в степень вводится вертикальная стрелка. Логические переменные и выражения в АЛГОЛе-60 обслуживаются знаками отношений $<$, $\leqslant$, $=$, $\neq$, $\geqslant$, $>$ и знаками логических действий $\vee$, $\wedge$, $\neg$, $\supset$, $\equiv$. Дополнительно по отношению к ФОРТРАНу используются квадратные скобки, открывающая и закры-

Таблица 3.2

<table>
<tr><th rowspan="2">Группа символов</th><th colspan="3">Языки программирования</th><th rowspan="2">Примечание</th></tr>
<tr><th>АЛГОЛ-60</th><th>Базисный ФОРТРАН</th><th>Подмножество ПЛ/I</th></tr>
<tr><td>Алфавит</td><td></td><td></td><td></td><td></td></tr>
<tr><td>латинский</td><td>от A до Z и от a до z</td><td>от A до Z</td><td>от A до Z</td><td></td></tr>
<tr><td>национальный</td><td>от А до Я и от а до я</td><td>—</td><td>—</td><td>В нашем случае—русский</td></tr>
<tr><td>Специальные символы</td><td>—</td><td>$</td><td>#, $, @</td><td></td></tr>
<tr><td>Цифры</td><td>от 0 до 9</td><td>От 0 до 9</td><td>От 0 до 9</td><td></td></tr>
<tr><td>Скобки</td><td>(|) | [|] | ‘ | ’</td><td>(|)</td><td>(|)|’ (апостроф)</td><td></td></tr>
<tr><td>Символы операций:</td><td></td><td></td><td></td><td></td></tr>
<tr><td>логических</td><td>∨|∧|¬|⊃|≡</td><td>—</td><td><u>|</u>|&|¬</td><td rowspan="3">Знак | для отличия от таких же знаков метаязыка здесь подчеркнут</td></tr>
<tr><td>арифметических</td><td>+|—|×|/|÷|↑</td><td>+|—|*|/|**</td><td>+|—|**|/|**</td></tr>
<tr><td>отношения</td><td><|≤|=|≠|≥|></td><td>—</td><td><|=|></td></tr>
<tr><td>Разделители</td><td>.|,|:|;|$_{10}$|</td><td>.|,|=|пробел</td><td>.|,|:|;|?|—| % пробел</td><td></td></tr>
</table>

вающая кавычки, двоеточие и точка с запятой. Для построения констант с плавающей точкой применяется подстрочная десятка $_{10}$.

В ПЛ/1, как и в ФОРТРАНе, используются только прописные латинские буквы, а из дополнительных буквенных знаков &, #, $, @. Для логического ИЛИ используется вертикальная черта |, две такие черты (||) обозначают знак сцепления строк. Знаки отношений конструируются из символов >,<,= и ¬, а знаки логических операций из символов |, & и ¬.

В ПЛ/1 используются только круглые скобки и одна универсальная кавычка — апостроф. Двоеточие и точка с запятой используются как в АЛГОЛе.

3.1.4. Имена. *Имена* (в АЛГОЛе-*идентификаторы*) присваиваются переменным, массивам (наборам переменных с одинаковыми свойствами), точкам входа в программу или подпрограмму и меткам (в ФОРТРАНе метки кодируются числами).

В АЛГОЛе и ФОРТРАНе употребляются только простые имена. Сложные имена, которые присваиваются в ПЛ/1 элементам структуры, состоят из простых имен, разделяемых точками, например:

<имя 1>·<имя 2>·<имя 3>

Простые имена или идентификаторы начинаются с буквы, за которой могут следовать один или несколько буквенно-цифровых символов, например

— для АЛГОЛА-60

A, C, AcX12, X1P2T3M4B5AAA

— для Базисного ФОРТРАНа, где число символов ограничено шестью:

A, C, ACX12, X1P2T3;

— наконец, для ПЛ/1, где число символов не должно превышать 31, а для «внешних» символов (с областью действия EXTERNAL) — 6, и где кроме буквенных и цифровых символов допускается знак подчеркивания:

A, C, ACX12, X1PAT3M8_B5_AABBCC

Сложные имена в ПЛ/1 могут иметь вид

A.ABC2_HP8.AA7, T17_CC2.ABC

Наиболее частые ошибки в написании имен состоят в использовании цифры в качестве первого символа, разделении символов простого имени пробелами (в АЛГОЛе такое разделение разрешено), точками, косой чертой и т. п., нарушении ограничений по числу символов имени.

Диагностический аппарат трансляторов ДОС/ЕС позволяет программисту быстро отыскать такие ошибки.

3.1.5. Константы. *Константами* в алгоритмических языках называются объекты, которые в процессе выполнения программы не изменяют своих значений. Синтаксис констант описан в табл. 3.3.

Известный алгольный способ представления арифметических констант можно записать в виде

[[+|—] [<целое без знака>] [[·] [<целое без знака>]]] [$_{10}$[+|—]<целое без знака>]

Таким образом, в АЛГОЛе-60 правильно написанными константами будут

+25, 37, —17, 37.5, $26.12_{10}2$, 0.26, 26, 312_{10}—3, 314159.2_{10}—5, $_{10}$+3, 1_{10}—6

Отнесение константы к числу *целых* или *действительных* производится на основе анализа ее написания. Так, 37 — целая, 37.5 — действительная, 37_{10}+3 — действительная константа, хотя в обычном понимании число 37_{10}+3=37000 — целое. Разграничение происходит не по факту целостности числа, а по способу представления.

Заметим, что в АЛГОЛе написание числа в виде 37. недопустимо. Чтобы отметить представление числа 37 как действительного, его надо написать, например, в виде $37_{10}0$. Обычно для реализаций АЛГОЛа-60 арифметические константы ограничены по длине 9 цифрами.

В ФОРТРАНе (в том числе в Базисном) целая константа изображается в виде

[+|—]<целое без знака>

и константа вида 8. или —3. уже не считается целой.

Действительная константа в этом языке имеет вид

[+|—] [<целое без знака>].[<целое без знака>] [E|D[+|—]<целое без знака>]

Буква E в константах обычной длины (когда константа занимает одно машинное слово) играет ту же роль, что и подстрочная десятка в АЛГОЛе. Для константы двойной длины (DOUBLE PRECISION) буква E заменяется на D. Все составляющие в представлении действительного числа, кроме десятичной точки, могут отсутствовать (но не одновременно).

Ниже приведены примеры действительных констант обычной длины в ФОРТРАНе

+2, 10.5, 2.7E4, .271828E+1

Таблица 3.3

<table>
<tr><th rowspan="2">Металингвистические переменные</th><th colspan="3">Языки программирования</th><th rowspan="2">Примечание</th></tr>
<tr><th>АЛГОЛ-60</th><th>ФОРТРАН</th><th>Подмножество ПЛ/1</th></tr>
<tr><td><константа></td><td><арифметическая константа>|<логическая константа></td><td><арифметическая константа></td><td><арифметическая константа>|<строчная константа></td><td></td></tr>
<tr><td><арифметическая константа></td><td><целая константа>|<действительная константа></td><td><целая константа>|<действительная константа></td><td><константа с фиксированной точкой>|<константа с плавающей точкой></td><td>В АЛГОЛе и ФОРТРАНе предполагаются десятичными</td></tr>
<tr><td><целая константа> (в ПЛ/1 — <константа с фиксированной точкой>)</td><td>[+|—]<целое без знака></td><td>[+|—]<целое без знака></td><td><целое двоичное число>B|<десятичное число></td><td></td></tr>
<tr><td><целое двоичное число></td><td>—</td><td>—</td><td><двоичная цифра>···</td><td></td></tr>
<tr><td><действительная константа> (в ПЛ/1 — <константа с плавающей точкой>)</td><td>[+|—][<десятичное число>][<десятичный порядок>]</td><td><действительная константа обычной длины>|<действительная константа двойной длины></td><td><двоичное число><двоичный порядок>B|[+|—]<десятичное число><десятичный порядок></td><td></td></tr>
<tr><td><действительная константа обычной длины></td><td>—</td><td>[+|—][<целое без знака>].[<целое без знака>].<десятичный порядок></td><td>—</td><td>Общее число цифр до и после точки не более 7</td></tr>
</table>

Продолжение табл. 3.3

Металингвистические переменные	Языки программирования			Примечание
	АЛГОЛ-60	ФОРТРАН	Подмножество ПЛ/1	
<действительная константа двойной длины>	—	[+|—][<целое без знака>]·[<целое без знака>] <десятичный порядок>	—	Общее число цифр до и после точки не более 17
<десятичное число>	[<целое без знака>] [<десятичная дробь>]	—	[<целое десятичное число>][·][<целое десятичное число>]	
<десятичная дробь>	·<целое без знака>	—	—	
<целое без знака> (в ПЛ/1—<целое десятичное число>)	<цифра> ···	<цифра> ···	<десятичная цифра>···	
<целое двоичное число>	—	—	<двоичная цифра> ···	

Продолжение табл. 3.3

Металингвистические переменные	Языки программирования			Примечание
	АЛГОЛ-60	ФОРТРАН	Подмножество ПЛ/1	
<десятичный порядок>	$_{10}$<целая константа>	[E|D]<целая константа>	E[+|—]<десятичное число>(не более чем двухзначное)	D[+]—в ФОРТРАНе для констант двойной длины
<двоичный порядок>	—	—	E[+|—]<десятичное число>(не более чем трехзначное)	
<двоичное число>	—	—	[<целое двоичное число>].[<целое двоичное число>]	
<логическая константа>	**true**|**false**	—	—	
<строчная константа>	—	—	[(n)]{<строка знаков>|<строка битов>}	
<строка знаков>	—	—	'<знак> ···'	
<строка битов>	—	—	'<двоичная цифра>···'B	

и примеры констант двойной длины

12345678.9876, 3141592653589.79D—12

Кроме арифметических констант, которые исчерпывают всё разнообразие констант Базисного ФОРТРАНа, в АЛГОЛе-60 имеются логические константы **true** и **false.**

Набор констант языка ПЛ/1 значительно шире.

Арифметические константы в ПЛ/1 делятся на константы с *фиксированной* точкой и константы с *плавающей* точкой. Каждая из них может быть *десятичной* или *двоичной*. Кроме арифметических, в ПЛ/1 имеются *строчные* константы. Они делятся на *символьные* и *битовые* константы. Последние являются обобщением логических констант АЛГОЛа.

Десятичную константу с фиксированной точкой можно представить в одном из двух видов:

[+|—]<целое десятичное число>[·]

или

[+|—] [<целое десятичное число>].<целое десятичное число>

Таким образом, десятичными константами с фиксированной точкой в ПЛ/1 являются

2419, —2419, +0.371, .371

Десятичные константы с плавающей точкой имеют вид

[+|—]<десятичное число><десятичный порядок>

где <десятичный порядок> пишется в виде

E[+|—]{<цифра>|<цифра><цифра>}

Буква Е в отличие от ФОРТРАНа используется для констант любой длины, допускаемой реализацией языка. Отсутствие квадратных скобок у метапеременной <десятичное число> в определении константы с плавающей точкой не позволяет в ПЛ/1 использовать аналог алгольной записи $_{10}+5$ и писать 1E+5. В остальном конструкция этих констант совпадает с алгольной.

Для двоичных констант с фиксированной точкой предполагается, что они являются целыми (точка фиксирована после последней цифры). Их вид

<целое двоичное число>B

Такими константами, например, являются

1B, 1001B, 11101001B, 1000000010B

Наконец, двоичные константы с плавающей точкой в ПЛ/1 представляются в виде

<двоичное число><двоичный порядок>B

где метапеременная <двоичный порядок> пишется в виде

E[+|—]{<цифра>|<цифра><цифра>|<цифра><цифра><цифра>}

и цифры предполагаются десятичными.

Таким образом, двоичные константы с плавающей точкой могут принимать, например, значения

10111E2B, 101E+21B, 11101E—217B

Символьные константы или строки знаков представляют собой последовательности знаков, заключенные в апострофы:

’A’, ’ABCD’, ’АБВГ’, ’A.12;/))&’

причем в них допустимы любые знаки, которые можно закодировать 256 комбинациями нулей и единиц байта. Если же в строке должен содержаться апостроф, то он удваивается. Так

’’’’, ’A’’B’’2’, ’AA’’A’’’’

означают соответственно ’, A’B’2 и AA’A’’. Обычное ограничение строки знаков — 256 символов.

Битовые константы или строки битов строятся из ограниченных апострофами последовательностей двоичных цифр 0 и 1, после которых ставится буква B. Так,

’0’B, ’1101’B, ’00010100’B

— битовые константы, тогда как

’0’, ’1101’, ’00010100’

— символьные.

Помещение перед строкой коэффициента повторения позволяет сократить написание строчных данных с повторениями. Так, строки (10)’A’ и (5)’1100’B соответственно эквивалентны строкам ’AAAAAAAAAA’ и ’11001100110011001100’B.

Строки, применяемые в спецификациях процедур АЛГОЛа или в операциях ввода — вывода ФОРТРАНа, не могут рассматриваться как константы, так как они не входят в состав выражений и не присваиваются в качестве значений каким-либо переменным.

Выражения в алгоритмических языках можно классифицировать по типу результата и по наличию самого низкоприоритетного знака операции, а в АЛГОЛе-60 еще и по наличию или отсутствию условия. Синтаксис выражений с указанием синонимов приведен в табл. 3.4.

Порядок выполнения операций в выражениях

↑ (или **)
× (или *) / и ÷, все три на равных правах,
+ и — (на равных правах),
|| (операция сцепления),
все операции сравнения на равных правах,
¬ (только в АЛГОЛе),
∧ (или &)
∨ (или |).

Операции возведения в степень наравне с одноместными операциями (+, —, ¬) в ПЛ/1 считаются наиболее приоритетными, а синтаксис языка обеспечивает выполнение их справа налево на равных правах. Справа налево выполняется операция возведения в степень и в ФОРТРАНе. В АЛГОЛе запись X↑Y↑Z означает $(X^Y)^Z$, т. е. X^{YZ}, тогда как в ФОРТРАНе и ПЛ/1 запись X**Y**Z означает привычное для нас X^{Y^Z}.

Язык ПЛ/1 оказывается очень удобным для обработки различных информационных массивов, в том числе и текстовых, из-за наличия в нем операции сцепления строк, возможности выделения подстрок, вмонтирования в строку в определенные позиции других строчных данных и пр. Такие операции допустимы не только с символьными, но и с битовыми строками. Это позволяет при более компактной символике реализовать на ПЛ/1 все возможности КОБОЛа.

Именующие выражения АЛГОЛа-60 разрешают оперировать с метками как с переменными: с помощью условного выражения выбирается по условию одна из двух меток, а с помощью переключателя — одна из нескольких. В ПЛ/1 аналогичные возможности дает введение переменных типа метки и присвоение им значений соответствующих констант.

В табл. 3.5 даны примеры эквивалентной записи выражений различных категорий в рассматриваемых язы-

Таблица 3.4

<table>
<tr><th rowspan="2">Выражение</th><th colspan="3">Языки программирования</th><th rowspan="2">Примечание</th></tr>
<tr><th>АЛГОЛ-60</th><th>Базисный ФОРТРАН</th><th>Подмножество ПЛ/1</th></tr>
<tr><td><выражение></td><td><простое выражение>| <условие> <простое выражение> else <выражение></td><td>—</td><td>—</td><td>Условные выражения используются только в АЛГОЛе-60</td></tr>
<tr><td><простое выражение></td><td><простое арифметическое выражение> | <простое булевское выражение> | <простое именующее выражение></td><td><простое арифметическое выражение></td><td>—</td><td>В ФОРТРАНе и ПЛ/1 просто <выражение></td></tr>
<tr><td><простое булевское выражение></td><td>[<простое булевское выражение>≡] <импликация></td><td>—</td><td>—</td><td>—</td></tr>
<tr><td><импликация></td><td>[<импликация> ⊃] <булевский терм></td><td>—</td><td>—</td><td>—</td></tr>
<tr><td><булевский терм> (он же—<выражение> в ПЛ/1)</td><td>[<булевский терм>∨] <булевский множитель></td><td></td><td>[<выражение>|]<выражение-6></td><td>—</td></tr>
<tr><td><булевский множитель> (выражение-6 в ПЛ/1)</td><td>[<булевский множитель>∧] <вторичное булевское выражение></td><td></td><td>[<выражение - 6>&] <выражение-5></td><td><выражение-5> в ПЛ/1 соответствует отношению в АЛГОЛе</td></tr>
</table>

Продолжение табл. 3.4

<table>
<tr><th rowspan="2">Выражение</th><th colspan="3">Языки программирования</th><th rowspan="2">Примечание</th></tr>
<tr><th>АЛГОЛ-60</th><th>Базисный ФОРТРАН</th><th>Подмножество ПЛ/1</th></tr>
<tr><td><вторичное булевское выражение></td><td>[¬]<первичное булевское выражение></td><td>—</td><td>—</td><td></td></tr>
<tr><td><первичное булевское выражение></td><td><логическое значение>| <логическая переменная> | <отношение> | <логическая функция>| (<булевское выражение>)</td><td>—</td><td>—</td><td></td></tr>
<tr><td><отношение>(<выражение-5> в ПЛ/1)</td><td><простое арифметическое выражение> <знак операции отношения> <простое арифметическое выражение></td><td>—</td><td>[<выражение-5>знак <операцни сравнения>] <выражение-4></td><td>В ПЛ/1 допускается цепочка типа a> >b=c≠d и т. п., так как операнды — строки, а результат сравнения тоже строка (битовая, длиной 1)</td></tr>
<tr><td><выражение-4></td><td>—</td><td>—</td><td>[< выражение-4> ‖] <выражение-3></td><td></td></tr>
</table>

<table>
<tr><th rowspan="2">Выражения</th><th colspan="3">Языки программирования</th><th rowspan="2">Примечание</th></tr>
<tr><th>АЛГОЛ-60</th><th>Базисный ФОРТРАН</th><th>Подмножество ПЛ/1</th></tr>
<tr><td><простое арифметическое выражение> (<выражение-3>ПЛ/1)</td><td>[<простое арифметическое выражение>] [+|—] <терм></td><td>[<простое арифметическое выражение>] [+|—]<терм></td><td>[<выражение-3> {+|—}] <выражение-2></td><td></td></tr>
<tr><td><терм> (<выражение-2> ПЛ/1)</td><td>[<терм> {×|/|÷}] <множитель></td><td>[<терм> {*|/}] <множитель></td><td>[<выражение-2> {*|/}] <выражение-1></td><td></td></tr>
<tr><td><множитель> ↑ (<выражение-1> ПЛ/1)</td><td>[<множитель> ↑] <первичное выражение></td><td><первичное выражение> [** <множитель>]</td><td><простое выражение> [** <выражение-1>] | {+|—|¬} <выражение-1></td><td>Перестановка метапеременных в ФОРТРАНе и ПЛ/1 обеспечивает выполнение операции ** справа налево</td></tr>
<tr><td><первичное выражение> („простое“ в ПЛ/1)</td><td><число без знака>| <переменная>|<функция> | (<арифметическое выражение>)</td><td><число без знака> | <переменная>| <функция | (<арифметическое выражение>)</td><td><имя>|<константа> |<вызов функции>| (<выражение>)</td><td></td></tr>
</table>

Продолжение табл. 3.4

Выражения	Языки программирования			Примечание
	АЛГОЛ-60	Базисный ФОРТРАН	Подмножество ПЛ/1	
<переменная> („имя" в ПЛ/1) *)	<идентификатор> | <идентификатор> [<список индексов>]	<идентификатор> [(<список индексов>)]	<неиндексированное имя> [(<список индексов>)]	
<неиндексированное имя>	—	—	<идентификатор> [.<идентификатор>]...	
<условие>	if <булевское выражение> then	—	—	
<простое именующее выражение>	<метка>|<указатель переключателя> | (<именующее выражение>)	Некоторый аналог представляет аппарат „вычисляемого" оператора GOTO	—	
<метка>	(см. табл. 3.9)			

*) В данной позиции для АЛГОЛа-60 квадратные скобки не символ метаязыка, а разделитель.

ках для случаев, когда их запись в этих языках имеет заметные различия.

Расширение набора логических операций в АЛГОЛе за счет введения знаков $\supset$ и $\equiv$, как известно, непринципиально, так как без них можно обойтись путем некоторого усложнения выражений. Полное же отсутствие логических выражений в Базисном ФОРТРАНе при решении некоторых задач оказывается неудобным, поэтому программистам, предпочитающим ФОРТРАН, можно посоветовать пользоваться имеющимся в последних версиях ДОС/ЕС ФОРТРАНом IV, где знаки $\wedge$, $\vee$, $\neg$ пишутся в виде .AND., .OR., .NO..

3.3. Описания

Если в программе встречается какое-либо имя и над ним необходимо произвести ту или иную операцию, то при трансляции программы необходимо знать, допустима ли эта операция. Такие сведения транслятор может получать из разных источников:

— иногда появление имени в определенном контексте уже полностью определяет его свойства, как, например, в АЛГОЛе текст

M1 : X : =X+1;

определяет M1 как метку, но если имя встречается в разных контекстах, то приписываемые ему свойства могут взаимно исключать друг друга. Такое неявное описание свойств имени иногда недостаточно;

— возможны предварительные соглашения, как в ФОРТРАНе, когда имена, начинающиеся с I, J, K, L, M, N, определяют целые переменные, а остальные—действительные. Аналогичное соглашение действует и в ПЛ/1;

— наконец, когда таких способов определения свойств имени недостаточно, применяют явное описание.

Описываются *переменные* (простые или неиндексированные), *массивы* (наборы переменных с одинаковыми свойствами), *функции* (процедуры-функции, подпрограммы-функции). В АЛГОЛе-60 описываются также переключатели и процедуры (как процедуры общего вида, так и процедуры-функции). В ПЛ/1 имеются также описания структур — наборов переменных с различными свойствами.

Таблица 3.5

<table>
<tr><th rowspan="2">Тип выражений</th><th colspan="3">Языки программирования</th></tr>
<tr><th>АЛГОЛ-60</th><th>Базисный ФОРТРАН</th><th>Подмножество ПЛ/1</th></tr>
<tr><td>Первичное выражение („простое“ в ПЛ/1)</td><td>(X↑Y)</td><td>(X ** Y)</td><td>(X**Y)</td></tr>
<tr><td>Множитель („выражение-1“ в ПЛ/1)</td><td>X↑Y</td><td>X**Y</td><td>X**Y</td></tr>
<tr><td></td><td>X↑Y↑Z</td><td>(X**Y)**Z</td><td>(X**Y)**Z</td></tr>
<tr><td></td><td>X↑(Y↑Z)</td><td>X**Y**Z</td><td>X**Y**Z</td></tr>
<tr><td>Терм („выражение-2“ в ПЛ/1)</td><td>2×A/B</td><td>2.*A/B</td><td>2*A/B</td></tr>
<tr><td></td><td>1÷3×L</td><td>1/3*L</td><td>1/3*L</td></tr>
<tr><td>Простое арифметическое выражение („выражение-3“ в ПЛ/1)</td><td>A+X/Y↑B</td><td>A+X/Y**B</td><td>A+X/Y**B</td></tr>
<tr><td></td><td>I—J÷K+L</td><td>I—J/K+L</td><td>I—J/K+L</td></tr>
<tr><td>Выражение-4 (ПЛ/1)</td><td>—</td><td>—</td><td>A+B||A—B/C</td></tr>
<tr><td>Отношение („выражение-5“ в ПЛ/1)</td><td>X>2
X=2↑Y</td><td>—</td><td>X>2
X=2**Y
X>2||2**Y</td></tr>
<tr><td>Первичное булевское выражение</td><td>true, X>2 (A=B∨C)</td><td>—</td><td>—</td></tr>
<tr><td>Вторичное булевское выражение</td><td>X>2
¬(X>2)</td><td>—</td><td>—</td></tr>
<tr><td>Булевский множитель („выражение-6“ в ПЛ/1)</td><td>A=B∧X>2
C≠D∧¬(X=Y)</td><td>—</td><td>A=B&X>2
C¬=D&¬(X=Y)</td></tr>
<tr><td>Булевский терм („выражение“ в ПЛ/1)</td><td>A=B∨X>2∨C
A∧C=D∨X=Y</td><td>—
—</td><td>A=B|X>2|C
A&C=D|X=Y</td></tr>
<tr><td>Импликация</td><td>A=B⊃A≥B</td><td>—</td><td>—</td></tr>
<tr><td>Простое булевское выражение</td><td>A>B≡B<A</td><td>—</td><td>—</td></tr>
</table>

Кроме того, в ПЛ/1 вводятся так называемые управляющие переменные, из которых в подмножестве ПЛ/1 используются данные типа *метка* и типа *указатель*. Управляющие переменные также могут быть простыми переменными или массивами.

3.3.1. Описание простых переменных. В АЛГОЛе-60 арифметические переменные делятся на целые — **integer** и действительные — **real**. Различия между десятичными и двоичными переменными не делается. Для логических переменных вводится описание **Boolean**.

Несколько особое положение занимает описатель **own** — собственный. Переменные с этим описателем за пределами области их определения не используются, но и не разрушаются, и при повторном выполнении входящей в эту область части программы учитываются значения этих переменных, полученные при предыдущем выполнении.

В ФОРТРАНе, когда неудобно пользоваться описанием по соглашению, возможно явное описание переменных операторами описания типа, имеющими вид

<тип> <список>

где <тип> — REAL, INTEGER или DOUBLE PRECISION, а список представляет собой перечень имен описываемых переменных, а также функций, разделенных запятыми (см. ниже). И если для REAL и INTEGER при соответствующем выборе первой буквы имени переменной наличие описания не обязательно, то для DOUBLE PRECISION оно необходимо всегда.

В ФОРТРАНе, как и в АЛГОЛе, не делается никаких различий между переменными, представляемыми в десятичной и в двоичной системах счисления, и в этом одно из отличий описания переменных в этих языках и в ПЛ/1.

Логических переменных в Базисном ФОРТРАНе нет.

В ПЛ/1 всякий набор явных описаний (объявлений) начинается с ключевого слова DECLARE (объявить), за которым следуют отдельные «частные» объявления, разделенные запятыми. Каждое частное объявление состоит из имени описываемого объекта и относящихся к нему описателей. Если между описателями нет никакого разделителя, то они отделяются от имени и друг от друга хотя бы одним пробелом.

Арифметические переменные могут иметь:

— описатель системы счисления: DECIMAL для десятичной и BINARY для двоичной;

— описатель формы представления: FIXED для переменных, представляемых в виде чисел с фиксированной точкой (в том числе целых переменных), и FLOAT для переменных, представляемых в виде чисел с плавающей точкой;

— описатель разрядности числа, отделяемый от остальных описателей скобками:

(n, m)

где n — общее число разрядов в описываемой переменной, а m — число дробных разрядов (если m=0, то это означает, что описываемая переменная — целая).

Пусть X — десятичная переменная, значения которой выражаются числами с тремя знаками до и двумя после запятой, Y — шестизначными десятичными числами в форме с плавающей точкой, а Z — целыми четырехзначными десятичными числами.

Описать эти переменные можно так:

```
DECLARE
   X DECIMAL FIXED (5, 2),
   Y DECIMAL FLOAT (6),
   Z DECIMAL FIXED (4);
```

Описания в ПЛ/1 считаются операторами и, подобно операторам АЛГОЛа, завершаются точкой с запятой. Возможность трактовать любое количество пробелов как один разделитель позволило использовать более наглядную запись: каждую переменную вместе с ее описателями выделить в отдельную строку.

Порядок следования описателей в ПЛ/1 произволен, но описатель разрядности не должен идти первым, так как иначе (см. ниже) он будет истолкован как размерность массива. Описатели можно выносить за скобки, вследствие чего X, Y, Z можно описать и в виде

```
   DECLARE(X, FIXED(5,2),Y FLOAT(6),Z
FIXED(4))DECIMAL;
```

Описание

```
   DECLARE((X, Y)FIXED,Z FLOAT)BINARY(23);
```

означает, что X, Y, Z — двоичные переменные, представляемые 23-разрядными двоичными числами, причем X и Y — целые, а Z — переменная с плавающей точкой.

Если один или несколько из описателей отсутствуют, то по умолчанию принято:

— при отсутствии описателя формы представления считать, что действует описатель FLOAT;

— при отсутствии описателя системы счисления считать действующим описатель DECIMAL;

— при отсутствии описателя разрядности полагать разрядность равной (5) для переменных DECIMAL FIXED, (15) для BINARY FIXED, (6) для DECIMAL FLOAT и (21) для BINARY FLOAT;

— при полном отсутствии описателей считать их BINARY FIXED(15) для имен, начинающихся с I, J, K, L, M, N, и DECIMAL FLOAT (6) для остальных имен.

Символьные строчные переменные в ПЛ/1 имеют описатель CHARACTER, а битовые — описатель BIT. В обоих случаях описатель длины (аналог описателя разрядности арифметических переменных) имеет вид (п) и указывает длину переменной в символах или битах. При отсутствии этого описателя длина переменной полагается равной 1.

Таким образом, оператор

DECLARE(A, B)CHARACTER(10),C BIT;

определяет 10-байтовые символьные переменные A и B и битовую переменную C длиной 1 бит.

В ПЛ/1 имеется еще способ описания переменных с помощью *шаблона* (PICTURE).

Для арифметических переменных с помощью шаблона можно определить количество позиций в представлении числа и способ их заполнения. В частности, цифровая позиция обозначается цифрой 9, а позиция, занимаемая десятичной точкой, — буквой V. Таким образом, оператор

DECLARE XYZ PICTURE 999V99';

определяет переменную, которая может иметь три знака до и два после десятичной точки. По аналогии с сокращенным представлением строчных констант это же можно записать с помощью коэффициента повторения в виде '(3)9V(2)9. Нули при описании цифровых позиций с помощью цифры 9 не гасятся и числа 23.12 и 0.17 будут соответственно представлены в виде 023.12 и 000.17. Заметим, что коэффициент повторения в спецификации шаблона находится внутри строки и ограничен апострофами, тогда как при написании строчной константы он помещается впереди апострофа.

Если незначащие нули требуется погасить, то вместо девятки в шаблоне используют букву Z. Использование вместо девятки знака * приводит к замене незначащих нулей звездочками. В этом случае позиция десятичной точки отмечается не буквой V, а точкой.

Для печати знака числа используется буква S, и тогда печатается как знак +, так и знак —. Написание в шаблоне перед цифровыми позициями знака + или — приводит к тому, что указанный знак выводится на печать, а противоположный заменяется пробелом.

Ниже приводится ряд примеров использования шаблона:

Исходное число	Шаблон	Результат печати	Исходное число	Шаблон	Результат печати
5376.1	9999V9	5376.1	+12.17	SZZZZ.ZZ	+ 12.17
5376.1	99V9	76.1	—12.17	SSSSS.ZZ	— 12.17
5376.1	9999V999	5376.100	—12.17	S****.ZZ	— **12.17

Наличие символа Y приводит к замене пробелом как значащего, так и незначащего нуля.

Шаблон используется и при описании строчных переменных. Здесь символ X означает любой символ, A — букву или пробел, 9 — цифру или пробел.

Арифметическими и строчными переменными в ПЛ/1 исчерпываются переменные, относящиеся к классу обрабатываемых. Как говорилось выше, в ПЛ/1 имеются еще управляющие переменные — переменные типа метка (LABEL) и переменные типа указатель (POINTER). Переменным типа метка присваиваются значения типа метка — идентификаторы. Это позволяет в ПЛ/1 выполнять функции, возлагаемые в АЛГОЛе на переключатель.

Переменные и константы типа указатель позволяют определить адреса расположения в памяти тех или иных объектов программы.

Область действия описания переменной определяется описателями EXTERNAL и INTERNAL, из которых второй приписывается по умолчанию именам простых переменных, массивов и структур, и поэтому явно почти никогда не указывается. Описатель EXTERNAL по умолчанию приписывается именам файлов, процедур и точек входа. Если же область действия переменной, массива,

структуры требуется расширить, то эти объекты следует объявить EXTERNAL.

3.3.2. Описание массивов. В АЛГОЛе-60 описателем массива служит слово **array**, перед которым могут быть описатели типа **integer** или **Boolean**. Сочетание **real array** хотя и допустимо, но практически не употребляется, так как синтаксис АЛГОЛа позволяет заменять его описателем **array** без указания типа.

Одним описателем **array** можно описать любое число массивов, для чего вслед за ним пишутся идентификаторы этих массивов, сопровождаемые списками граничных пар. Массивы, имеющие одинаковые списки граничных пар, объединяются в сегменты — их идентификаторы пишутся друг за другом через запятые, а за последним из них ставится общий список граничных пар.

Граничная пара — это совокупность двух выражений, разделенных двоеточием. Первое выражение определяет нижнюю границу области, которую может пробегать индекс, а второе — верхнюю границу. Количество граничных пар в АЛГОЛе не ограничено.

Как и простые переменные, массивы могут иметь описатель **own**.

Пусть необходимо описать состояние твердого тела в некоторый момент t, задав его координаты в прямоугольной системе координат Oxyz, составляющие вектора скорости вдоль этих же осей, три угла Эйлера, характеризующие положение некоторой системы координат $O'\xi\eta\zeta$, связанной с телом, относительно системы Oxyz, и производные этих углов по времени. Совокупность переменных

$$t, x, y, z, v_x, v_y, v_z, \varphi, \psi, \theta, \varphi', \psi', \theta'$$

мы можем объявить массивом из 13 элементов

array D[0 : 12];

и тогда под $D[0]$ будем понимать t, под $D[5]$ — v_y и т. д.

Зафиксировав состояние тела в моменты t_1, t_2, … …, t_N, можно рассматривать их как двумерный массив — матрицу

array D2[0 : 12, 1 : N];

На выражения, определяющие границы, не накладывается ограничения, чтобы они определяли целые значения. Округление выполняет транслятор, а для массивов, размерность которых определяется в процессе выполне-

ния программы, — сама программа. Единственное требование состоит в том, чтобы нижняя граница не превышала верхней.

В ФОРТРАНе основная задача описания массива — задание его р а з м е р н о с т и. При неявном описании типа массива размерность его указывается в операторе DIMENSION, записываемом в виде

DIMENSION <список описаний>

а каждое из описаний, отделяемое от предыдущего запятой, имеет вид

<имя массива>(<список верхних границ>)

Нижняя граница всегда предполагается равной 1. Количество верхних границ не должно превышать трех, и выражаться они должны целыми константами без знака. Динамические массивы в ФОРТРАНе не предусмотрены.

Приведенный выше первый из примеров на ФОРТРАНе получил бы описание вида

DIMENSION D(13)

а для второго переменную N следовало бы заменить какой-либо целой константой без знака.

Если массив описывается явно, то вместо оператора DIMENSION ставится его тип, а структура списка верхних границ остается прежней. Наш массив D можно было бы описать оператором

REAL D(13)

Аналогично можно было бы написать

INTEGER A(2, 6),B(10),C
DOUBLE PRECISION P(6)

что означало бы введение в программу матрицы целых переменных A_{ij} размером 2×6, вектора целых переменных B_k с десятью составляющими и целого скаляра C, а также 6-мерного вектора P, переменных с удвоенным количеством разрядов.

В ФОРТРАНе имеется возможность с помощью оператора COMMON определить общую область, в которой будут одновременно располагаться простые переменные и/или массивы из разных сегментов программы. И если размерность массива не указана ни в операторе DIMENSION, ни в явном описании, то это можно сделать в операторе COMMON:

COMMON <список описаний>

причем строение этого списка такое же, как и в предыдущих описаниях массивов.

Описание массивов в ПЛ/1 отличается от описания скалярных переменных только наличием между именем массива и описателями его элементов описателя размерности, который для реализованного в ДОС/ЕС подмножества языка по возможностям совпадает с описателем размерности в Базисном ФОРТРАНе—размерность массивов не превышает трех, а в качестве верхних границ допускаются только целые константы без знака.

Рассмотренный выше пример на ПЛ/1 можно записать в виде

DECLARE D(13), DECIMAL FLOAT;

При объявлении многих массивов одинаковой размерности ее можно выносить за скобки, например

DECLARE(X, Y, Z) (2, 6)CHARACTER(20);

означает введение трех матриц 2×6, элементы которых— 20-символьные строки.

3.3.3. Описание структур. *Структура* — это совокупность данных, которые не могут быть охарактеризованы одним набором описателей. Подобных объектов нет в ФОРТРАНе и АЛГОЛе, они попали в ПЛ/1 из КОБОЛа, где именуются *записями*. В виде структур могут быть представлены различные списки, формуляры, накладные, квитанции, анкеты и пр.

Основной признак структуры — ее иерархичность. Самой структуре присваивается уровень 1, а ее элементам или частям — какой-либо уровень n>1. Сами эти части могут быть простыми переменными, массивами и структурами. В последнем случае они считаются подструктурами. Подструктуры имеют аналогичное строение, но уровень их элементов m должен быть больше n. На практике обычно уровни нумеруются последовательными натуральными числами.

Примером структуры могут быть данные о железнодорожном билете

```
DECLARE
  1 БИЛЕТ,
    2 МАРШРУТ,
      3 ОТ CHARACTER(15),
      3 ДО CHARACTER(15),
    2 КАССА PICTURE'(3)9',
    2 ПОЯС PICTURE'(3)9',
```

```
2 ОТПРАВЛЕНИЕ,
  3 ДАТА,
    4 ЧИСЛО PICTURE'99',
    4 МЕСЯЦ PICTURE'99',
    4 ГОД PICTURE'99',
  3 ВРЕМЯ,
    4 ЧАСЫ PICTURE'99',
    4 МИНУТЫ PICTURE'99',
2 ПОЕЗД,
  3 НОМЕР PICTURE'99',
  3 ВИД PICTURE'9',
2 ВАГОН,
  3 НОМЕР PICTURE'99',
  3 ТИП PICTURE'99',
2 МЕСТО PICTURE'99',
2 ЦЕНА,
  3 РУБЛЕЙ PICTURE'999',
  3 КОПЕЕК PICTURE'99';
```

В этом примере элементами структуры БИЛЕТ оказались простые переменные КАССА, ПОЯС и МЕСТО и подструктуры МАРШРУТ, ОТПРАВЛЕНИЕ, ПОЕЗД, ВАГОН и ЦЕНА. Подструктура ОТПРАВЛЕНИЕ, в свою очередь, состоит из подструктур ДАТА и ВРЕМЯ, которые представляют собой наборы простых переменных ЧИСЛО, МЕСЯЦ, ГОД, ЧАСЫ и МИНУТЫ.

В подструктурах ПОЕЗД и ВАГОН имеются элементы с одинаковыми именами НОМЕР, но уточненные их имена различны. В первом случае это

БИЛЕТ.ПОЕЗД.НОМЕР

а во втором

БИЛЕТ.ВАГОН.НОМЕР

Если один из элементов структуры — массив, указание конкретной единицы данных в структуре требует знания индексов элемента массива.

Пусть структура А объявлена оператором

```
DECLARE
  1 A,
    2 B(10, 3)FIXED DECIMAL,
    2 C CHARACTER(15);
```

Чтобы указать элемент третьей строки и второго столбца матрицы B, его можно написать в виде A.B(3,2) или A(3,2).B или даже A(3).B(2). Важен только порядок и наличие всех индексов, а к какому из частных имен они приписаны, не имеет значения.

3.3.4. Прочие описания. Кроме изложенных, изучавшим АЛГОЛ знакомы еще описания переключателей и описания процедур.

В ФОРТРАНе аналогом переключателя является так называемый «вычисляемый» оператор GOTO, применение которого не требует специальных описаний. В ПЛ/1 переключатель заменяется введением переменных типа метка, значения которых могут поступать через канал ввода и потом присваиваться переменной в нужном месте программы.

Что касается процедур в ФОРТРАНе и внешних процедур в ПЛ/1, то они рассматриваются как программные единицы и имеют статус, почти равный статусу главной программы. Их тексты находятся за пределами главной программы и не могут рассматриваться как описания.

В ПЛ/1 объектами описания могут быть еще имена точек входа в программы и имена файлов, рассматриваемые в гл. 4.

3.3.5. Классы памяти. Основной выигрыш, получаемый в языке программирования от введения блочной структуры программы, — экономия памяти. Так, в ФОРТРАНе, где блочная структура синтаксисом языка не предусмотрена, каждая объявленная (явно или неявно) переменная или массив получает необходимую область памяти в процессе трансляции программы. В АЛГОЛе же транслятор строит программу так, что на этапе трансляции память выделяется только тем объектам, которые объявляются в самом старшем блоке. Для остальных объектов память выделяется не транслятором, а самой программой в момент передачи управления тому блоку, в котором эти объекты объявлены. Это, естественно, несколько усложняет программу, но позволяет значительно экономить память. Эти два вида выделения участков памяти получили наименование статического и автоматического.

В языке ПЛ/1, где, как и в АЛГОЛе, программа может иметь блочную структуру, эти два вида выделения памяти также используются и при объявлении объектов могут включаться в число его описателей как STATIC и AUTOMATIC, но обычно они явно среди описателей не

присутствуют, так как STATIC присваивается объектам, объявление которых имеет область действия EXTERNAL, а AUTOMATIC — объектам с областью действия INTERNAL.

Имеется еще и третий вид выделения памяти для объектов языка ПЛ/1: память выделяется не транслятором и не программой, а самим программистом. Объявляя тот или иной объект, программист может указать, что ему отводится тот же участок памяти, что и другому, ранее объявленному объекту. В этом случае среди описателей объекта должен присутствовать описатель DEFINED с указанием объекта, для которого уже выделена память одним из ранее перечисленных способов. Такой способ объявления объектов называется *переопределением*. Не разрешается в ПЛ/1 использовать переопределение по отношению к объектам, которым память выделена с помощью переопределения.

Кроме переопределения, программист может выделить память для объекта с помощью аппарата *указателей*, имеющегося в языке ПЛ/1. Указатель — это переменная, значением которой является адрес какого-либо объекта. Переменные, получающие участок памяти с помощью указателей, называются *базированными* и получают описатель BASED. Объект, чья память предоставляется для базированного объекта, называется его *базой*. В качестве базы в ПЛ/1 не могут быть использованы объекты, получившие место в памяти с помощью базирования.

Синтаксис описаний приводится в табл. 3.6.

3.4. Простые операторы

В синтаксисе АЛГОЛа-60 деление операторов на простые и более сложные ведется с помощью отсечения сначала условного оператора, потом оператора цикла и далее блока и составного оператора. В результате таких отсечений остаются только операторы, не содержащие внутри себя других операторов, которые и именуются простыми или по терминологии АЛГОЛа-60 основными.

В АЛГОЛе-60 к таким операторам относятся операторы присваивания, перехода, вызова процедуры и пустой оператор. Операторы ввода—вывода в АЛГОЛе-60 являются частными случаями оператора вызова процедуры.

Таблица 3.6

<table>
<tr><th rowspan="2">Металингвистические переменные</th><th colspan="3">Языки программирования</th></tr>
<tr><th>АЛГОЛ-60</th><th>Базисный ФОРТРАН</th><th>Подмножество ПЛ/1</th></tr>
<tr><td><объявление> (в АЛГОЛе-60—<описание>)</td><td><описание типа>|<описание массива>|<описание переключателя>|<описание процедуры></td><td><объявление переменной>|<объявление массива></td><td>DECLARE [<частное объявление>] ...</td></tr>
<tr><td><частное объявление></td><td>—</td><td>—</td><td><частное объявление переменной>|<частное объявление массива>|<частное объявление структуры></td></tr>
<tr><td><описание типа></td><td>[<область действия>] <тип><список имен></td><td>—</td><td>—</td></tr>
<tr><td><объявление переменной> (в ПЛ/1—<частное объявление переменной>)</td><td>—</td><td><неявное объявление>|<тип><список имен></td><td><неявное объявление>|<имя><пробел>{<описатель>}...</td></tr>
<tr><td><область действия></td><td>own</td><td>—</td><td>EXTERNAL | INTERNAL</td></tr>
<tr><td><тип></td><td><арифметический тип>|<логический тип></td><td><арифметический тип></td><td>—</td></tr>
<tr><td><арифметический тип></td><td>real | integer</td><td>REAL | INTEGER | DOUBLE PRECISION</td><td>—</td></tr>
<tr><td><логический тип></td><td>Boolean</td><td>—</td><td>—</td></tr>
</table>

Продолжение табл. 3.6

Металингвистические переменные	Языки программирования		
	АЛГОЛ-60	Базисный ФОРТРАН	Подмножество ПЛ/1
<имя>	<идентификатор>	<идеитификатор>	(см. табл. 3.7)
<описание массива>	[<область действия>] [<тип>] **array** <список массивов>	—	—
<объявление массива> (в ПЛ/1—<частное объявлеиие массива>)	—	<иеявное объявление> | <ключевое слово> <список массивов>	<имя> <размерность>{<описатель>} ...
<описатель>	—	—	<способ представления>|<система счисления>|<длина>|<разрядность>|<область действия>|<описатель строчных объектов>|<описатель управляющих объектов>|<описатель класса памяти>
<система счисления>	—	—	DECIMAL | BINARY (по умолчанию DECIMAL)
<способ представления>	—	—	FIXED | FLOAT (по умолчанию FLOAT)

Продолжение табл. 3.6

Металингвистические переменные	Языки программирования		
	АЛГОЛ-60	Базисный ФОРТРАН	Подмножество ПЛ/1
<описатель строчных объектов>	—	—	CHARACTER | BIT | PICTURE
<описатель управляющих объектов>	—	—	LABEL | POINTER
<описатель класса памяти>	—	—	STATIC | AUTOMATIC
<разрядность>	—	—	(<натуральное число>) | (<натуральное число>, <целое число>)
<длина>	—	—	(<натуральное число>)
<неявное объявление>	—	Переменные, имена которых начинаются с I, J, K, L, M, N, считаются целыми, остальные — действительными	Переменные, имена которых начинаются с I, J, K, L, M, N, считаются BINARY FIXED (15), остальные — DECIMAL FLOAT (6)
<ключевое слово>	—	<тип> | DIMENSION | COMMON	—

Продолжение табл. 3.6

Металингвистические переменные	Языки программирования		
	АЛГОЛ-60	Базисный ФОРТРАН	Подмножество ПЛ/1
<список массивов>	<сегмент массива> | <список массивов>, <сегмент массива>	<имя><размерность> | <список массивов>, <имя><размерность>	—
<сегмент массива>	<имя><размерность> | <сегмент массива>, <имя>, <размерность>	—	В ПЛ/1 конструкция, аналогичная сегменту АЛГОЛа, строится путем выноса описателей за скобки
<размерность>*)	[<список граничных пар>]	(<список верхних границ>)	(<список верхних границ>)
<список граничных пар>	<граничная пара> | <список граничных пар>, <граничная пара>	—	—
<граничная пара>	<нижняя граница>: <верхняя граница>	—	—
<список верхних границ>	—	<верхняя граница> | <список верхних границ>, <верхняя граница>	<верхняя граница> | <список верхних границ>, <верхняя граница>
<верхняя (нижняя) граница>	арифметическое выражение	<целое число>	<целое число>

Продолжение табл. 3.6

Металингвистические переменные	Языки программирования		
	АЛГОЛ-60	Базисный ФОРТРАН	Подмножество ПЛ/1
<описание переключателя>	**switch** <имя>:=<переключательный список>	—	—
<переключательный список>	<именующее выражение>|<переключательный список>,<именующее выражение>	—	—
<частное объявление структуры>	—	—	<номер уровня><имя> [<описатель размещения>] <список объявлений компонентов>
<список объявлений компонентов>	—	—	<номер уровня><частное объявление>|<список объявлений компонентов>, <номер уровня><частное объявление>
<номер уровня>	—	—	<натуральное число> (1 для структуры, уровень вложенности для компонентов)
<описатель размещения>	—	—	ALIGHNED | UNALIGHNED

*) У етапеременной <список граничных пар> квадратные скобки — разделитель АЛГОЛа.

Набор простых операторов Базисного ФОРТРАНа примерно совпадает с алгольным. Что же касается ПЛ/1, то здесь прямой аналогии провести нельзя, так как в этом языке существуют простые операторы, не имеющие самостоятельного смысла за пределами содержащих их сложных операторов (групп, блоков, процедур). Эти операторы подобны ключевым словам АЛГОЛа.

Ниже простые операторы рассматриваются друг за другом в традиционном порядке.

3.4.1. Оператор присваивания. Возможности оператора присваивания АЛГОЛа-60 в наибольшей мере иллюстрируются примером

begin real x; **integer** i; **Boolean** B; ... ...; i : = x : = **if** B **then** 3.51 **else** 5.2;
end

Здесь производится групповое присваивание значений i и x, в процессе присвоения значения i действительные числа преобразуются в целые; правая часть оператора в этом примере — условное выражение. Заметим попутно, что перестановка в левой части оператора ... x : = i : = ... привела бы к x=i, что не имеет места в приведенном выше примере: при B=**true** мы получим x=3.51, а i=4; если же B=**false**, то получается x=5,2; i=5.

Оператор присваивания в ФОРТРАНе допускает только однократное присваивание. Одновременно он производит необходимые преобразования величин. Так, можно написать

```
X=3,51
X=5,2
I=3,51
I=5,2
```

и эти операторы дадут те же результаты, что и в приведенном примере на АЛГОЛе. Но программисту необходимо организовать переходы к нужным в определенной ситуации операторам, для чего потребуется применение операторов перехода.

В языке ПЛ/1, в полном объеме реализованном в ОС/ЕС, допускается групповое присваивание:

A, B, C=0;

но в его подмножестве, реализованном в ДОС/ЕС, такая возможность отсутствует. Большим расширением

возможностей оператора присваивания в ПЛ/1 является присваивание значений массиву и структуре. Имеется также возможность присваивать значения и отдельным элементам как массива, так и структуры.

Пусть A, B, C — матрицы

$$\begin{pmatrix}1&0&1\\0&1&0\\0&0&1\end{pmatrix}, \begin{pmatrix}1&0&0\\1&0&0\\1&0&0\end{pmatrix}, \begin{pmatrix}1&1&1\\0&0&0\\0&0&0\end{pmatrix},$$

D, E, F — векторы (1, 0, 0), (0, 1, 0), (0, 0, 1), X — матрица 3×3, Y — трехмерный вектор.

Применяя операторы присваивания

```
X=B+C—A+1;
Y=5*D—E+2*F+3;
```

получаем

$$X=\begin{pmatrix}2&2&2\\2&0&1\\2&1&0\end{pmatrix}, \quad Y=\begin{pmatrix}8\\2\\5\end{pmatrix}$$

Правые части таких операторов являются выражениями типа массив. Так, единица в первом из них — матрица 3×3, все элементы которой единицы, а тройка во втором — вектор, все составляющие которого тройки.

Оператор присваивания значений структуре в правой части должен содержать также выражение типа *структура*, которое имело бы смысл. Обычно эта структура-константа.

Структуре, описанной в п. 3.3.3, можно присвоить значение, задаваемое, например, следующими потоками символов:

```
К И Ш И Н Е В               М
О С К В А                 0 5
0 0 7 0 2 4 0 8 7 8 1 6 4 5 0 0 4
8 1 0 3 3 2 8 0 2 0 7 0
```

Возможны и частичные присвоения:

```
ЦЕНА='02070'; ВИД='1';
```

и т. д., если имя, стоящее в левой части оператора, уникально в данной программной единице. Но для имени НОМЕР, как говорилось выше, необходимо употребить уточненное имя

```
БИЛЕТ.ПОЕЗД.НОМЕР='048';
```

или для данного случая достаточно

ПОЕЗД.НОМЕР='048';

Возможны и такие частичные присвоения:

ДО=ДО ||' — КИЕВСКАЯ';

после чего станцией назначения будет уже не 'МОСКВА', а 'МОСКВА — КИЕВСКАЯ';

МАРШРУТ='КИШИНЕВ'||(8)' '||'МОСКВА';

В последнем случае 30-байтовое поле подструктуры МАРШРУТ заполнится слева 21 байтом строчного выражения правой части оператора. Правые 9 байтов останутся пустыми.

В табл. 3.7 приводится синтаксис оператора присваивания.

3.4.2. Оператор перехода. В АЛГОЛе-60 оператор перехода — один из простейших операторов.

Его вид

goto <именующее выражение>;

Он может быть проще или сложнее только за счет изменения вида именующего выражения. Последнее же (см. § 3.2) бывает простым и условным, а простое именующее выражение — либо меткой, либо указателем переключателя. Таким образом, с помощью одного оператора перехода можно передать управление в любое число точек программы. Так, если описан переключатель

switch A : =A1,A2,A3,A4;

то за счет управления значениями переменной B и параметра k можно заставить оператор

goto if B>0 **then** D **else** A[k];

передать управление как на D, так и на любую из меток A1—A4.

В базисном ФОРТРАНе имеются две формы оператора перехода: безусловный GOTO вида GOTO m и вычисляемый GOTO

GOTO (m_1, m_2, ..., m_k), I

В безусловном операторе GOTO m (не более чем пятизначное число) — метка, стоящая перед одним из операторов, которому и передается управление.

Например, программа

A=0.
B=0.

Таблица 3.7

Металингвистические переменные	Языки программирования		
	АЛГОЛ-60	Базисный ФОРТРАН	Подмножество ПЛ/1
<оператор присваивания>	<список левой частн> <правая часть>	<левая часть><правая часть>	<левая часть><правая часть>
<список левой части>	<левая часть>|<список левой части><левая часть>	—	—
<левая часть>	<переменная>:=	<перемениая>=	<имя>=
<правая часть>	<арифметическое выражение>|<булевское выражение>	<арифметическое выражение>	<выражение>
<имя>	—	—	<простое имя>|<имя>.<простое ими>
<простое имя>	—	—	<идентификатор>|<идентификатор>(<список индексов>)

```
10  B=B+0.1
    B=B * B
    A=A+B
    GOTO 10
```

будет до бесконечности вычислять сумму квадратов чисел 0.1, 0.2, ...

В вычисляемом операторе GOTO m_1, m_2, ..., m_k — метки операторов, которым передается управление, когда управляющая переменная I принимает значения 1, 2, ..., k.

Используем вычисляемый GOTO, чтобы написанная выше программа могла прервать вычисления. Для этого перепишем ее в виде

```
    A=0.
    B=0.
    I=0.
10  B=B+0.1
    B=B * B
    A=A+B
    I=I+1
    GOTO(10,10,10,10,20),I
20  END
```

Теперь конечное значение A окажется равным 0.01+ +0.04+0.09+0.16+0.25, после чего I станет равным 5 и управление перейдет не на метку 10, как в предыдущих проходах, а на метку 20 (конец программы).

В ПЛ/1 оператор перехода имеет вид

GOTO <имя>;

или

GO TO <имя>;

причем имя может быть как простым, так и индексированным и во втором случае управление будет передано одному из элементов массива меток. Имя бывает и составным, так как метка или массив меток может быть элементом структуры.

Программа предыдущего примера на ПЛ/1 может иметь вид

```
DECLARE M(2) LABEL;
  A=0; B=0; M(1)='C'; M(2)='D';
  C : B=B+0.1; A=A+B * B;
      GOTO M(1+B/10);
  D : END;
```

Суммирование квадратов чисел 0.1, 0.2, ... происходит до тех пор, пока округление значения 1+B/10 будет давать единицу, т. е. до B=0.5. Выбрав делитель другим, можно прервать суммирование на другом количестве повторений.

Синтаксис оператора перехода дается в табл. 3.8.

3.4.3. Другие простые операторы. Кроме рассмотренных выше, в АЛГОЛе-60 к простым операторам относятся оператор процедуры (точнее — оператор вызова процедуры), пустой оператор и операторы ввода — вывода. В ФОРТРАНе простыми являются условный оператор, оператор цикла и операторы паузы и остановки процесса выполнения программы. В ПЛ/1 имеется значительное количество операторов синтаксически простых, но не применяющихся за пределами более сложных конструкций (групп, блоков, процедур). Поэтому в дальнейшем:

— условные операторы, циклы, блоки и процедуры излагаются для всех языков совместно независимо от простоты или сложности этих операторов в том или ином языке;

— операторы для более сложных конструкций, рассматриваются вместе с этими конструкциями;

— операторы вызова процедур рассматриваются вместе с процедурами.

Таким образом, в данном разделе осталось рассмотреть только пустой оператор.

В АЛГОЛе он является средством обхода части программы с выходом на конец блока, цикла, составного оператора. Пустой оператор в АЛГОЛе не употребляется без меток, которые в данном случае предшествуют слову **end** или точке с запятой.

В ФОРТРАНе имеются ограничения на использование некоторых операторов как последних в цикле. Не могут заканчивать цикл условный оператор, оператор перехода, вложенный оператор цикла. Для передачи управления в конец цикла, заканчивающегося одним из этих операторов, тело цикла дополняют пустым оператором CONTINUE. Не выполняя никаких операций, он, имея перед собой метку, может принять на себя управление и в зависимости от исчерпания списка цикла возвратить управление на его начало или осуществить выход из цикла.

В ПЛ/1 пустой оператор имеет ту же форму и играет ту же роль, что и в АЛГОЛе.

Таблица 3.8

Металингвистические переменные	Языки программирования		
	АЛГОЛ-60	Базисный ФОРТРАН	Подмножество ПЛ/1
<оператор перехода>	**goto** <именующее выражение>	<безусловный GOTO>|<вычисляемый GOTO>	GO TO <имя>|GOTO <имя>
<безусловный GOTO>	—	GOTO <целое без знака>*)	—
<вычисляемый GOTO>	—	GOTO (<список меток>), <целая неиндексированная переменная>	—
<список меток>	—	<целое без знака>|<список меток>, <целое без знака>	

*) Не более чем пятизначное число.

3.5. Условный оператор

Условный оператор — простейшее средство разветвления процесса обработки данных, но его возможности в значительной степени зависят от синтаксиса языка. Так, в ФОРТРАНе, где этот оператор следует отнести к простым, его использование часто невозможно без оператора перехода. В АЛГОЛе, где его конструкция более сложна, оператор перехода не всегда необходим, но имеются ограничения на использование условных операторов непосредственно после проверки условия. Наконец, в ПЛ/1 за счет введения дополнительных синтаксических правил снимается и это ограничение, но появляются дополнительные пустые операторы.

Вид условного оператора в АЛГОЛе-60

<условие> <безусловный оператор> [**else** <оператор>];

или

<условие> <оператор цикла>;

где условие имеет вид

if <логическое выражение> **then**

Вернемся теперь к нашему примеру вычисления суммы квадратов чисел 0.1, 0.2, ..., который на ФОРТРАНе без использования условного оператора выглядел искусственно. На АЛГОЛе он будет иметь вид

```
begin real A,B;
              A:=B:=0;
     M:B:=B+0.1;
              A:=A+B↑2;
              if B<1.9999 then goto M
end
```

Этот отрезок программы обеспечит вычисление суммы квадратов чисел от 0.1 до 2.0 через 0.1. Константа 1.9999 в логическом выражении условного оператора взята несколько меньше числа 2, но больше 1.9, чтобы приближенное представление действительного числа В не привело к добавлению к А лишнего слагаемого 2.1^2.

Во избежание этого обычно предпочитают пользоваться целыми числами:

```
begin real A; integer B;
              A:=B:=0;
```

```
M: B: =B+1;
            A: =A+B↑2/100;
         if B<20 then goto M
end
```

Применение полной формы условного оператора можно проиллюстрировать на примере одновременного вычисления суммы квадратов четных и нечетных чисел

```
begin
  integer A,B,C;
    A: = B: =C: =0;
  M: B: =B+1;
                 if B=B÷2×2 then A: =A+B↑2
                 else C: C+B↑2;
                 if B<20 then goto M
end
```

При делении нечетных чисел на два отбрасыванием остатка и последующем умножении также на два в первом из условных операторов условие B=B÷2×2 не выполняется, вследствие чего сумма их квадратов накапливается в C. Сумма же квадатов четных чисел, для которых упомянутое условие выполняется, будет накапливаться в A.

Использование условного оператора совместно с оператором цикла излагается в § 3.6.

Вид условного оператора в Базисном ФОРТРАНе

IF(<арифметическое выражение>)m_1, m_2, m_3

Здесь m_1, m_2, m_3 — метки операторов, на которые передается управление; как и в операторе перехода это целые без знака.

Изложенный выше пример с помощью оператора IF можно записать так:

```
  I=0
  J=0
  K=0
4 J=J+1
  IF(J/2*2—J)2,1,1
1 I=I+J*J
  GOTO 3
2 K=K+J*J
3 IF (J—20)4,5,5
5 Продолжение программы.
```

Переменная J определена как целая и деление ее на целое число 2 по синтаксису ФОРТРАНа приводит к целому результату. Нечетные числа J=2n+1 при делении их на два с последующим удвоением дают четные числа 2n, в скобках окажется отрицательное число —1, и управление пойдет на первую метку. При четном числе результат в скобках будет нулевым.

Условный оператор в ПЛ/1 имеет формы, подобные алгольным, но между условием и ключевым словом ELSE разрешается использовать условный оператор, причем только в его полной форме (со словом ELSE). В этом случае не возникает двухсмысленностей, которые могли бы возникнуть, если бы в этом месте условного оператора АЛГОЛа было разрешено использование условного оператора. Это достигается, между прочим, и тем, что в отличие от АЛГОЛа в ПЛ/1 нет операторов, которые бы не заканчивались символом ';'. В данном случае символ ';' ставится перед ELSE.

Рассмотрим простейший пример

```
C := 0;
if A then
   if B then C := 1
else C := 2;
```

Здесь **else** может быть отнесено к любому **if**, поэтому, если взять внутренний оператор в операторные скобки

```
C : 0;
if A then
begin if B then C := 1
end
else C := 2;
```

то в зависимости от значений A и B переменная C принимает значения

B	A	
	true	**false**
true	1	2
false	0	2

тогда как при другой расстановке скобок

```
    C := 0;
if A then
begin
    if B then C := 1 else C := 2
end
```

мы будем иметь

B	A	
	true	**false**
true	1	0
false	2	0

Для ПЛ/1 при первой расстановке алгольных скобок эта программа имеет вид

```
C=0;
IF A THEN
   IF B THEN C=1; ELSE;
ELSE C=2;
```

а при второй

```
C=0;
IF A THEN
   IF B THEN C=1; ELSE C=2;
```

Каждое слово ELSE «закрывает скобку», открываемую ближайшим слева IF.

Между IF и THEN и между THEN и ELSE или после THEN, когда за ним не следует ELSE, может размещаться и несколько операторов, оформленных в группу или обычный блок.

Синтаксис условного оператора приведен в табл. 3.9.

3.6. Составной оператор (простая группа)

Образований, подобных составному оператору АЛГОЛа, в ФОРТРАНе нет. В ПЛ/1 аналогом составного оператора является так называемая *простая группа*.

Таблица 3.9

<table>
<tr><th rowspan="2">Металингвистические переменные</th><th colspan="3">Языки программирования</th></tr>
<tr><th>АЛГОЛ-60</th><th>Базисный ФОРТРАН</th><th>Подмножество ПЛ/1</th></tr>
<tr><td><условный оператор></td><td><условие> <безусловный оператор> | <условие> <оператор цикла> | <условие> <безусловный оператор> else <оператор></td><td><условие> <метка> <метка> [, <метка>]</td><td><условие> <блок-1> | <условие> <вставка IF> ELSE <блок-1> | <условие> <блок-1> ELSE <блок-1></td></tr>
<tr><td><условие></td><td>if <булевское выражение> then</td><td>IF (<арифметическое выражение>)</td><td>IF <выражение> THEN</td></tr>
<tr><td><метка></td><td><идентификатор></td><td><целое без знака></td><td><идентификатор></td></tr>
<tr><td><блок-1></td><td>—</td><td>—</td><td><оператор> | <группа> | <обычный блок></td></tr>
<tr><td><вставка IF></td><td>—</td><td>—</td><td><условие> <вставка IF> ELSE <блок-2></td></tr>
<tr><td><блок-2></td><td>—</td><td>—</td><td><вставка IF> | <безусловный оператор> | <группа> <обычный блок></td></tr>
</table>

В АЛГОЛе-60 составной оператор представляет собой объединение нескольких операторов (в числе которых могут в свою очередь быть и составные), ограниченное операторными скобками **begin** и **end.** Описаний внутри составного оператора нет.

В ПЛ/1 операторными скобками, ограничивающими группу, являются операторы DO и END. Между ними могут находиться операторы, группы, обычные и процедурные блоки и даже описания DECLARE и FORMAT. Но наличие описаний не делает группу блоком (в смысле АЛГОЛа), так как вводимые в группе имена могут использоваться и вне ее в минимальном охватывающем эту группу обычном или процедурном блоке. Такое невозможно в АЛГОЛе, где все описания блока должны предшествовать первому из операторов («выполняемых» операторов ФОРТРАНа и ПЛ/1).

Программа на языке АЛГОЛ

```
begin real x, y; integer i, n;
      i := 0; x := 1; y := 0; n := 2;
   M : if i<n then
   begin i := i+1;
      y := y+x↑i; goto M
   end
i := 0; n := n+1; if n<10 then goto M
end
```

в ПЛ/1 может принять вид

```
A : PROCEDURE OPTIONS (MAIN);
    DECLARE
    Y FLOAT DECIMAL,
    (I, N) FIXED BINARY;
    I=0; X=1; Y=0; N=2;
M : IF I<N THEN DO; I=I+1;
    Y=Y+X**I;
    DECLARE X DECIMAL FLOAT;
    GOTO M; END;
    I=0; N=N+1; IF N<10 THEN
    GOTO M;
END A;
```

Пропущенное описание программист АЛГОЛа вынужден вставлять в начало блока, а в ПЛ/1 этого ограничения нет. Можно внутрь группы поместить и процедурный блок, который будет иметь силу и за пределами группы, внутри охватывающего его обычного или про-

цедурного блока. На выполнение группы процедурный блок не влияет, так как всегда обходится программой и может быть активизирован только оператором вызова процедуры.

Синтаксис простой группы приводится в табл. 3.10.

3.7. Оператор цикла (итеративная группа)

Многократный возврат к определенному месту программы можно организовать и с помощью условного оператора в сочетании с оператором перехода. Но программисту значительно удобнее применять для этого *оператор цикла.*

С помощью оператора цикла легко записывать перебор, изменение параметра с постоянным или переменным шагом, повторение алгоритма до выполнения логического условия.

В АЛГОЛе-60 оператор цикла состоит из *заголовка цикла* и *тела цикла* — оператора, подлежащего выполнению. Этот оператор может быть сколь угодно сложным за счет использования средств объединения простых операторов в составные и в блоки.

Заголовок цикла в АЛГОЛе

for <параметр цикла> := <список цикла> **do**

в качестве параметра цикла может иметь как простую, так и индексированную переменную целого и действительного типа. Элементы списка, разделенные запятыми, в простейшем случае могут быть константами или арифметическими выражениями. Более сложные элементы списка могут организовывать перебор значений параметра с некоторым постоянным или переменным шагом

<арифметическое выражение> **step** <арифметическое выражение> **until** <арифметическое выражение>

пока параметр не достигнет определенного конечного значения, которое в процессе выполнения цикла также может меняться. Наконец, в АЛГОЛе допускаются элементы списка вида

<арифметическое выражение> **while** <булевское выражение>

Таблица 3.10

<table>
<tr><th rowspan="2">Металингвистические переменные</th><th colspan="2">Языки программирования</th></tr>
<tr><th>АЛГОЛ-60</th><th>Подмножество ПЛ/1</th></tr>
<tr><td><составной оператор> (<простая группа> ПЛ/1)</td><td>begin <конец составного></td><td>DO; <конец группы></td></tr>
<tr><td><конец составного> (<конец группы> в ПЛ/1)</td><td><оператор> end | <оператор>; <конец составного></td><td><предложение> END; | <предложение> <конец группы></td></tr>
<tr><td><предложение></td><td>—</td><td><описание> | <оператор> | <группа> | <начальный блок> | <процедурный блок></td></tr>
<tr><td><блок> (<начальный блок> ПЛ/1)</td><td><начало блока> <конец составного></td><td>BEGIN; {<предложение>}...END;</td></tr>
</table>

Примечание. В ФОРТРАНе IV и в Базисном ФОРТРАНе составных операторов и блоков нет.

Примером оператора цикла, содержащего элементы всех трех типов, может служить оператор

for P : =0,1,3,7,P×2 **while** P<25,28 **step** 1 **until** 30 **do**
begin A : =A+P;B : =B+P↑2;C : =C+P↑3 **end**;

Здесь 0,1,3 и 7 — элементы первого типа, 28 **step** 1 **until** 30 — элементы второго типа, а P×2 **while** P<25 — элементы третьего типа. Всего в этом цикле перебирается 8 значений P : 0, 1, 3, 7, 14, 28, 29 и 30.

Циклы в АЛГОЛе могут содержать младшие циклы, и глубина вложения может быть любой, но младший цикл должен полностью содержаться в операторе, следующем за **do** старшего цикла.

Например, для массива

array A [0 : 5, 2 : 15, 1 : 3, 0 : 6];

можно организовать цикл, подсчитывающий сумму значений его элементов с помощью оператора:

```
begin B : =0;
    for i : =0 step 1 until 5 do
    for j : =2 step 1 until 15 do
      for k : =1 step 1 until 3 do
        for l : =0 step 1 until 6 do
          B : =B+A[i,j,k,l]
end
```

Все написанное — один оператор АЛГОЛа, и поэтому никаких разделителей здесь не нужно. Раньше всех пробегает свои значения l, после чего изменяется значение k, а l пробегает свои значения вновь. До изменения значения i очередь доходит, когда пробежит все свои значения j.

В ФОРТРАНе оператор цикла называется оператором DO и имеет вид

DO<целое без знака><целая переменная>=m_1, m_2[,m_3]

причем m_i — натуральные числа. Например,

DO 25 M=10,20,2

Следующее за DO после пробела число — метка последнего из операторов, входящих в тело цикла, целая переменная — параметр цикла, m_1 — его начальное, m_2 — конечное значения, а m_3 — шаг изменения. Величину шага можно не указывать, если она равна единице.

Кроме натуральных чисел, величины m_i не могут принимать никаких других значений. В теле цикла нельзя изменять параметр цикла.

Оператор цикла ФОРТРАНа может иметь так называемую *расширенную область*, находящуюся за пределами тела цикла. Передача управления из тела цикла в расширенную область должна происходить с помощью операторов GOTO или IF, и таким же образом должен быть организован возврат из нее в тело цикла. В расширенной области не должно быть операторов DO. В ней также нельзя изменять параметр цикла.

Последний из операторов тела цикла не должен быть оператором DO, IF, GOTO. Поэтому в случае необходимости тело цикла дополняют пустым оператором CONTINUE.

Приведем несколько искусственный пример вычисления корней квадратного уравнения с помощью оператора цикла ФОРТРАНа:

```
      DIMENSION A(10), B(10), C(10), X1(10), X2(10),
                      RE(10), REAL IM(10)
        DO 5 I=1,10
              E=2.*A(I)
              D=B(I)*B(I)—4.*A(I)*C(I)
                 GOTO 3
1             X1(I)=(—B(I)+SQR(D))/E
               X2(I)=(—B(I)—SQRT(D))/E
                 GOTO 5
2             RE(I)=—B(I)/E
              IM(I)=SQRT(—D)/E
                 GOTO 5
3             IF(D) 2, 1, 1
5             CONTINUE
```

В нашем алгоритме IF — последний непустой оператор. Если бы вычисления велись один раз, то меткой 5 был бы отмечен первый из операторов продолжения программы. Но условный оператор не может быть последним среди операторов тела цикла, поэтому появилась необходимость в пустом операторе.

По своей структуре оператор цикла в ПЛ/1 очень похож на аналогичный оператор в АЛГОЛе. В соответствии со своим наименованием (итеративной группы) этот оператор состоит из итеративного оператора-DO

(аналог заголовка цикла в АЛГОЛе) и набора предложений, заканчивающихся оператором-END (аналог тела цикла АЛГОЛа). Оператор-END вида

END;

играет роль указателя последнего оператора тела цикла в ФОРТРАНе.

Итеративный оператор-DO имеет вид

DO <спецификация-DO>;

или

DO WHILE(<выражение>);

Слово DO играет роль слова **for**, а спецификация-DO — роль заголовка цикла в АЛГОЛе.

В подмножестве ПЛ/1, реализованном в ДОС/ЕС, параметр цикла обязательно простая переменная. Далее после знака присваивания (=) идет спецификация или набор спецификаций следующих видов:

<выражение>
<выражение> BY <выражение>
<выражение> TO <выражение>
<выражение> BY <выражение> TO <выражение>
<выражение> TO <выражение> BY <выражение>
<выражение> WHILE (<выражение>)

Во всех спецификациях, кроме последней, может присутствовать окончание WHILE (<выражение>).

В ПЛ/1 пропуск части спецификации BY <выражение> по аналогии с ФОРТРАНом производится тогда, когда <выражение> тождественно равно единице. Пропуск TO <выражение> предполагает, что цикл выполняется до тех пор, пока какой-либо из операторов тела цикла не передаст управление за его пределы.

Такой набор спецификаций перекрывает все виды элементов списка цикла АЛГОЛа.

Синтаксис оператора цикла (итеративной группы) дан в табл. 3.11.

Относительно вида предложений, которые могут входить в тело цикла в языке ПЛ/1, следует заметить, что в частном случае каждое из них может быть оператором, описанием, группой (в том числе итеративной),

Таблица 3.11

<table>
<tr><th rowspan="2">Металингвистические переменные</th><th colspan="3">Языки программирования</th></tr>
<tr><th>АЛГОЛ-60</th><th>Базисный ФОРТРАН</th><th>Подмножество ПЛ/1</th></tr>
<tr><td><оператор цикла></td><td><заголовок цикла> <тело цикла></td><td><заголовок цикла> <список цикла></td><td><итеративный оператор DO> <тело цикла> END;</td></tr>
<tr><td><заголовок цикла> (<итеративный оператор DO> в ПЛ/1)</td><td>for <параметр цикла>: = <список цикла> do</td><td>DO <метка конца цикла> <параметр цикла> =</td><td>DO <параметр цикла>=<список цикла> | DO WHILE (<выражение>)</td></tr>
<tr><td><параметр цикла></td><td><переменная></td><td><целая неиндексированная переменная></td><td><целая неиндексированная переменная></td></tr>
<tr><td><список цикла></td><td><элемент списка цикла> | <список цикла>, <элемент списка цикла></td><td><элемент списка цикла></td><td><элемент списка цикла> | <список цикла>, <элемент списка цикла></td></tr>
<tr><td><элемент списка цикла></td><td><арифметическое выражение> [step <арифметическое выражение> until <арифметическое выражение>] | <арифметическое выражение> while <булевское выражение></td><td><начальное значение>, <конечное значение> [,<шаг>]</td><td><выражение> [BY <выражение> [TO <выражение>] | TO <выражение> [BY <выражение>]] [WHILE (<выражение>)]</td></tr>
<tr><td><тело цикла></td><td><оператор></td><td>—</td><td rowspan="2"><предложение> | <список предложений> <предложение> <оператор> | <описание> | <группа> | <начальный блок> | <процедурный блок></td></tr>
<tr><td><предложение></td><td>—</td><td>—</td></tr>
</table>

обычным или процедурным блоком. Что касается описаний и процедурных блоков, то здесь справедливы все те замечания, которые были сделаны относительно структуры простой группы.

Приведем пример цикла, в котором используем по возможности все виды спецификаций ПЛ/1.

Пусть A_{ij} — матрица целых чисел, $i=1, 2, \ldots, m$, $j=1, 2, \ldots, n$ — натуральные числа, а B — выражение, истинность которого имеет место для простых и только для простых A_{ij}. Требуется построить сумму C квадратов всех простых A_{ij}, расположенных в четных строках матрицы 100×200.

На ПЛ/1 этот пример описывается отрезком программы

```
DECLARE A(100, 200), FIXED DECIMAL,
          C FIXED DECIMAL;
          C=0;
 DO J=1 TO 200;
   DO I=2 BY 2 TO 100;
          IF B THEN C=C+A (I, J)** 2;
   END;
 END;
```

3.8. Обычный (начальный) блок

Понятие *блока* впервые появилось в АЛГОЛе-60 и с тех пор используется в алгоритмических языках как средство экономии памяти при операциях с локальными переменными.

В ПЛ/1 роль блока играет обычный или начальный блок (блок BEGIN), получивший это название для отличия от процедурного блока.

Характерная особенность всякого блока — наличие в нем описаний объектов, не имеющих смысла или имеющих другой смысл за его пределами. Наличие описаний заставляет в АЛГОЛе ввести запрет на передачу управления извне блока иначе как через его начало. Это объясняется тем, что в соответствии с синтаксисом АЛГОЛа требуется поместить описания в начале блока, а при передаче управления в середину блока описания обходятся и программа может использовать некоторые переменные не в том смысле, как они должны пониматься внутри блока.

Память, выделенная для объектов блока, освобождается при его завершении, за исключением переменных с описателем **own** в АЛГОЛе и EXTERNAL в ПЛ/1.

В АЛГОЛе блок — один из видов представления программы. В ПЛ/1 же обычному блоку отведена более скромная роль: он никогда не может быть внешним.

Вхождение младших блоков в старшие называется *гнездованием блоков.* В ПЛ/1 при учете глубины гнездования принимаются во внимание как обычные, так и процедурные блоки. В подмножестве ПЛ/1, реализованном в ДОС/ЕС, глубина гнездования ограничена уровнем 3. В АЛГОЛе-60 ограничений на глубину гнездования нет.

Среди предложений, входящих в тело блока, могут быть описания (для АЛГОЛа также описания процедур), операторы, группы (составные операторы), блоки (для ПЛ/1 обычные и процедурные). В АЛГОЛе-60, как уже упоминалось, все описания должны предшествовать операторам. В ПЛ/1 это ограничеиие отсутствует.

Синтаксис блока (начального блока ПЛ/1) рассматривается в табл. 3.10 вместе с синтаксисом составного оператора.

3.9. Процедуры (подпрограммы)

Часть текста программы, в которой излагается содержание процедур, в АЛГОЛе-60 относится к описаниям, т. е. по терминологии ФОРТРАНа и ПЛ/1 — к невыполняемым операторам. Вместе с тем, воспринимая этот текст, транслятор переводит тело процедуры в такие же машинные коды, в какие переводятся и операторы, не входящие в тело процедуры. Поэтому относить процедуры к описаниям имеет смысл только потому, что при выполнении программы их обходят, а управление им передается дистанционно с помощью *оператора процедуры.*

Но точно такое же положение имеет место во всех версиях ФОРТРАНа и ПЛ/1. Процедуры при выполнении программы обходятся и обращаются к ним только с помощью *оператора вызова процедуры.* Тем не менее в этих языках процедуры (подпрограммы) относятся к «программным единицам», т. е. к группам операторов, среди которых обязательно присутствуют выполняемые.

Существенное же различие состоит только в том, что

процедуры в АЛГОЛе всегда находятся внутри некоторого блока, не являющегося процедурой (обычного блока по терминологии ПЛ/1). Это обстоятельство затрудняет в АЛГОЛе сборку программы из процедур, так как в этом языке процедура никогда не представляет собой готовую к выполнению программу. В ПЛ/1 и даже в предшественнике АЛГОЛа ФОРТРАНе процедура может быть внешним блоком, поэтому описанных трудностей не возникает.

Ниже наряду с процедурами в обычном смысле рассматривается также их частный случай — процедуры-функции.

Отдельно будут рассмотрены заголовок процедуры, ее тело и механизм активизации. Синтаксис процедур описывается в табл. 3.12.

3.9.1. Заголовок процедуры. В заголовке процедуры ФОРТРАНа и ПЛ/1 имеются ключевое слово, имя процедуры и список передаваемых процедуре и возвращаемых ею параметров. Если это процедура-функция, то в заголовке должна быть информация о типе ее значения. В АЛГОЛе-60 в отличие от этого тип и ключевое слово **procedure** в заголовок не входят, а последний имеет вид

<имя>[(<список параметров>)];[<список значений>;][<совокупность спецификаций>;]

Тип процедуры указывается только для процедур-функций и относится к имени процедуры. Возможны типы **real**, **integer** и **Boolean**.

Допускаются процедуры без параметров, и тогда в заголовке отсутствует список параметров. Примером этого могут быть обычные записи функции sinx, lnx и т. д., обычно запрещаемые в АЛГОЛе. Но если в программу включены описания процедур без параметров с такими именами, то программист может ими пользоваться.

Пусть имеется, например, программа

```
begin real x; ...
  real procedure sinx; sinx:=sin(x);
  real procedure lnx; lnx:=ln(x);
  . . . . . . . . . . . . . .
end
```

Таблица 3.12

Металингвистические переменные	Языки программирования			Примечание
	АЛГОЛ-60	Базисный ФОРТРАН	Подмножество ПЛ/1	
<подпрограмма> (<процедура> в ПЛ/1)	<внутренняя подпрограмма>	<внутренняя подпрограмма> | <внешняя подпрограмма>	<внутренняя процедура> | <внешняя процедура>	В языке ПЛ/1 допускаются как внутренние процедуры по типу АЛГОЛа, так и внешние (по типу ФОРТРАНа)
<внутренняя подпрограмма> (<внутренняя процедура> в ПЛ/1)	<описание процедуры>	<оператор-функция>	<процедурный блок>	
<внешняя подпрограмма (<внешняя процедура> в ПЛ/1)	—	<подпрограмма-функция> | <подпрограмма общего вида>	<процедурный блок>	
<оператор-функция>	—	<имя> (<список параметров>) = <выражение>	—	Помещается в разделе описаний
<описание процедуры> (<процедурный блок> в ПЛ/1)	[<тип`] **procedure** <заголовок процедуры> <тело процедуры>	—	<оператор-PROCEDURE><тело процедуры> <оператор-END>	

Металингвистические переменные	Языки программирования			Примечание
	АЛГОЛ-60	Базисный ФОРТРАН	Подмножество ПЛ/1	
<подпрограмма-функция>	—	<заголовок процедуры-функции> <тело процедуры> END	—	
<подпрограмма общего вида>	—	<заголовок процедуры общего вида> <тело процедуры> END	—	
<оператор-PROCEDURE> в ПЛ/1	—	—	<имя входа> : PROCEDURE (<список параметров>) [<описатели функции>];	
<заголовок процедуры>	<идентификатор процедуры> [(<список формальных параметров>)]; [<список значений>;] [<совокупность спецификаций>;]	—	—	
<заголовок процедуры-фукции> (для ФОРТРАНа)	—	<тип> FUNCTION <имя> (<список параметров>)	—	
<заголовок процедуры общего вида> (для ФОРТРАНа)	—	SUBROUTINE <имя> (<список параметров>)	—	

Продолжение табл. 3.12

<table>
<tr><th rowspan="2">Металингвистические переменные</th><th colspan="3">Языки программирования</th><th rowspan="2">Примечание</th></tr>
<tr><th>АЛГОЛ-60</th><th>Базисный ФОРТРАН</th><th>Подмножество ПЛ/1</th></tr>
<tr><td><список параметров> (в АЛГОЛе „формальных параметров“)</td><td><параметр> [,<параметр>] ··· | <пусто></td><td><параметр> [,<параметр>] ··· | <пусто></td><td><параметр> [,параметр>] ··· | <пусто></td><td>Во всех языках разрешаются процедуры без параметров</td></tr>
<tr><td><описатели функции> (<тип> в АЛГОЛе)</td><td>Аналогично типу простой переменной или массива</td><td>—</td><td><описатели арифметических данных>| <описатели строчных данных></td><td></td></tr>
<tr><td><список значений></td><td>value <список идентификаторов>; |<пусто></td><td>—</td><td>—</td><td></td></tr>
<tr><td><совокупность спецификаций></td><td><тип объекта> <список идентификаторов>; |<пусто></td><td>—</td><td>В ПЛ/1 спецификации АЛГОЛа заменяются в случае надобности описанием объектов</td><td></td></tr>
<tr><td><тип объекта></td><td>string | label | switch | [<тип>] array | [<тип>] procedure</td><td>—</td><td>—</td><td></td></tr>
<tr><td><тело процедуры></td><td><оператор></td><td>Совокупность операторов, заканчивающихся оператором END</td><td><предложение> | <оператор-ENTRY> | простой оператор-DO > < тело процедуры> <оператор-END></td><td></td></tr>
</table>

В такой программе можно пользоваться выражениями $1+\sin x$, $a+\sin x+b\times\ln x$, $\ln(\sin x)$ и т. д. Но $\sin y$, $\ln z$ по-прежнему останутся запрещенными.

Известный в АЛГОЛе способ передачи параметров по значениям и по наименованиям применяется программистом по желанию. Спецификации же, по форме напоминающие описания, в некоторых реализациях АЛГОЛа обязательны. Они служат транслятору указаниями о способах построения областей памяти для процедуры.

В ФОРТРАНе процедуры, вырабатывающие результат из одного значения, которое присваивается имени процедуры, называются функциями, а остальные процедуры именуются *подпрограммами*, что отмечается в первом случае ключевым словом FUNCTION, а во втором — SUBROUTINE.

Вид заголовка процедуры для этих двух случаев

[<тип>] FUNCTION <имя>](<список параметров>)]

и

SUBROUTINE <имя>[(<список параметров>)]

Процедуры без параметров в ФОРТРАНе применяются при использовании общей области, организуемой с помощью оператора COMMON, который должен присутствовать в этом случае как в вызывающей, так и в вызываемой процедуре.

Тип функции не указывается, если его можно определить по первой букве ее имени, как это делается и для простых переменных. Если же этим приемом воспользоваться нельзя или если необходимо ввести функцию типа DOUBLE PRECISION, то ее тип указывается перед ключевым словом FUNCTION. В левой части оператора присваивания в теле процедуры в этом случае должно хотя бы один раз появиться имя функции.

Кроме подпрограмм-функций, представляющих в ФОРТРАНе отдельные программные единицы, в этом языке есть и некоторое подобие процедур-функций АЛГОЛа; это так называемые операторы-функции. Располагаются они вместе с описаниями и имеют вид

<имя>(<список аргументов>) = <выражение>

В выражении такой функции нельзя использовать переменные с индексами, но можно применять указатели функций этого же типа, описание которых в программе предшествует описанию функции, их использующей.

Тип функции, введенной оператором-функцией, как и при неявном задании типа переменных и массивов, определяется по первой букве ее имени. Допускается для оператора-функции и явное описание типа.

В ПЛ/1 описательная часть процедуры называется оператором-PROCEDURE и имеет вид

<имя входа> : PROCEDURE[(<список параметров>)][<описатели функции>];

<имя входа> здесь заменяет идентификатор процедуры АЛГОЛа или подпрограммы ФОРТРАНа, но ставится оно перед ключевым словом PROCEDURE в качестве метки. Процедура в ПЛ/1 также может быть без параметров, и в этом случае она использует переменные с описателем EXTERNAL.

Описатели процедур-функций, служащие для определения типа значения, вырабатываемого такой процедурой, помещаются в конце оператора-PROCEDURE, а не перед именем функции, так как в ПЛ/1 оно вынесено в метку.

3.9.2. Тело процедуры. В АЛГОЛе-60 тело процедуры — либо оператор, либо машинный код. Машинный код часто применяется при построении процедур ввода — вывода. Если тело процедуры представляет собой блок, то в нем могут содержаться описания других процедур и обращения как к другим процедурам, так и к самой себе. Обращение к самой себе и сама процедура, в теле которой есть такое обращение, называются *рекурсивными*. Классический всюду приводящийся пример рекурсивной процедуры — вычисление факториала:

```
integer procedure factorial (n);
factorial := if n=0 then 1 else n×factorial (n—1);
```

В теле процедуры, состоящем из одного оператора присваивания, но с условным выражением, имеется обращение к той же процедуре. Выполнение такой процедуры, скажем, для n=5 приведет к вызову ее для n=4, 3,

2, 1 и 0 и только после этого ее выполнение будет продолжено в обратном порядке — сначала для n=0, потом для n=1, 2, 3, 4 и, наконец, для n=5.

Тело подпрограммы ФОРТРАНа включает в себя всю последовательность операторов, следующих за описательной частью; оно заканчивается оператором END. В отличие от ПЛ/1, где оператор END имеет более широкое применение, в ФОРТРАНе он только заключает программные единицы — программу и подпрограммы.

В ПЛ/1 тело процедуры может представлять собой набор предложений, т. е. описаний, операторов, групп и блоков, операторов-ENTRY и конструкций вида

DO;<тело процедуры>END;

Оператор-ENTRY — новая по сравнению с АЛГОЛом и ФОРТРАНом конструкция, допускающая использование не всей процедуры, а какой-либо ее части. Вид его подобен оператору-PROCEDURE:

<имя входа> : ENTRY[(<список параметров>)]
[<описатели функции>];

Заметим, что количество параметров оператора ENTRY может не совпадать с количеством параметров основного входа процедуры.

Пусть процедура A преобразует комплексное число Z=X+iY из обычной формы в тригонометрическую: Z=R(cos φ+isin φ), а процедуры B и C служат для определения отдельно модуля R числа Z и его аргумента φ:

```
A : PROCEDURE (X, Y, R, FI);
    I=0; GOTO M;
B : ENTRY (X, Y, R);
    I=1;
M : R=SQRT (X*X+Y*Y);
    IF I=1 THEN GOTO N;
C : ENTRY (X, Y, FI);
    FI=ATAN (Y/X);
N : END;
```

При выполнении процедуры A вычисляются и R и FI, так как локальная переменная I принимает нулевое значение. Оператор присваивания I=1 обходится, что обеспечивает вычисление FI. При входе же через B, наоборот, оператор I=1 работает и вызывает обход вычисления FI.

Возврат к месту вызова из тела процедуры ФОРТРАНа и ПЛ/1 обеспечивается оператором RETURN. В ФОРТРАНе он не содержит параметров, а в ПЛ/1 имеет вид

RETURN[(<выражение>)];

3.9.3. Механизм активизации процедур. Активизация процедур-функций осуществляется с помощью употребления в выражениях *указателя функции,* т. е. имени функции, сопровождаемого (в скобках) списком фактических параметров, если они требуются. Фактические параметры могут быть выражениями, значения которых (или программы вычисления которых при вызове параметров по наименованию) подставляются в тело процедуры-функции, обеспечивая вычисление ее значения. Различий в способах активизации процедур-функций в рассматриваемых языках нет.

Активизация процедур общего вида в АЛГОЛе-60 достигается использованием оператора процедуры вида

<имя>[(<список фактических параметров>)];

В ФОРТРАНе аналогичный оператор называется оператором вызова *подпрограммы* или оператором CALL. Его вид

<имя>[(<список фактических параметров>)];

В ПЛ/1 оператор вызова процедуры дополняется знаком «;», а именем может быть как имя самой процедуры, так и имя любого ее входа.

В области действия каждого входного имени оно должно быть уникальным. Это требование относится как к именам процедур, так и к именам дополнительных входов. Кроме того, имена процедур-функций без параметров не должны совпадать с именем какой-либо простой переменной, а имена функций с параметрами — с именами массивов (исключение составляет АЛГОЛ-60, где для индексных выражений используются квадратные скобки).

3.10. Библиотечные функции

Набор функций библиотеки АЛГОЛа, встроенной в транслятор, ограничен синусом, косинусом, арктангенсом, экспонентой, натуральным логарифмом, квадратным корнем, абсолютной величиной, целой частью и знаком

аргумента. Тригонометрические функции используют значения аргумента в радианной мере.

Библиотека Базисного ФОРТРАНа состоит из подпрограмм математических функций и служебных программ, которые здесь не рассматриваются.

Математические функции ФОРТРАНа могут использовать как аргументы обычного вида (действительные), так и аргументы типа DOUBLE PRECISION. Соответственно и результат определяется либо с обычным, либо с удвоенным числом разрядов. Перечень математических функций ФОРТРАНа и их характеристики приводится в табл. 3.13. По сравнению с АЛГОЛом в библиотеке ФОРТРАНа дополнительно имеются функции тангенса, гиперболического тангенса, десятичного логарифма, максимума и минимума группы чисел и остатка от деления целого на целое. Имеются также программы возведения чисел в любую степень, выраженную действительным числом или числом с удвоенным количеством разрядов.

Библиотека ПЛ/1 для его подмножества, реализованного в ДОС/ЕС, значительно богаче и состоит из нескольких подбиблиотек. Основные ее подбиблиотеки: математические функции, «арифметические» функции и функции со строчными аргументами.

Подбиблиотека математических функций содержит квадратный корень, экспоненту, натуральный, десятичный и двоичный логарифмы, круговые и гиперболические синус, косинус и тангенс и обратные функции — круговой и гиперболический арктангенсы и интеграл вероятностей. Круговые функции имеются в двух вариантах — для аргументов, выраженных в радианах и в градусах. При обращении к круговому арктангенсу результат может быть получен как в градусах, так и в радианах.

Подбиблиотека арифметических функций, имеющая несколько условное наименование, позволяет производить преобразования аргументов к двоичной или десятичной системе счисления, представлять их в форме с фиксированной или плавающей точкой, округлять до ближайшего меньшего или большего целого числа, сохранять заданное количество значащих цифр и т. п. В этой подбиблиотеке имеются функции, определяющие абсолютную величину аргумента, его знак, минимум или максимум группы аргументов. Аргумент, подвергающий-

Таблица 3.13

Содержание программы	Вид обращения	Значение функции	Количество аргументов	Тип аргумента	Тип функции
Показательная функция	EXP (X)	e^x	1	Д	Д
	DEXP (X)	"	1	ДР	ДР
Натуральный логарифм	ALOG (X)	$\ln x$	1	Д	Д
	DLOG (X)	"	1	ДР	ДР
Десятичный логарифм	ALOG10 (X)	$\log_{10} x$	1	Д	Д
	DLOG10 (X)	"	1	ДР	ДР
Арктангенс	ATAN (X)	arctg x	1	Д	Д
	DATAN (X)	"	1	ДР	ДР
Синус	SIN (X)	$\sin x$	1	Д	Д
	DSIN (X)	"	1	ДР	ДР
Косинус	COS (X)	$\cos x$	1	Д	Д
	DCOS (X)	"	1	ДР	ДР
Квадратный корень	SQRT (X)	$\sqrt{x}$	1	Д	Д
	DSQRT (X)	"	1	ДР	ДР
Гиперболический тангенс	TANH (X)	th x	1	Д	Д
	DTANH (X)	"	1	ДР	ДР
Целая часть числа	INT (X)	$[x]$	1	Д	Ц
	AINT (X)	"	1	Д	Д
	IDINT (X)	"	1	ДР	Ц
Максимальное значение	AMAX0 ($X_1,\ldots,X_k$)	Максимальное из k чисел $x_1, x_2, \ldots, x_k$	⩾2	Ц	Д
	AMAX1 ($X_1,\ldots,X_k$)		"	Д	Д
	MAX0 ($X_1,\ldots,X_k$)		"	Ц	Ц
	MAX1 ($X_1,\ldots,X_k$)		2	Д	Ц
	DMAX1 ($X_1,\ldots,X_k$)		2	ДР	ДР
Минимальное значение	AMIN0 ($X_1,\ldots,X_k$)	Минимальное из k чисел $x_1, x_2, \ldots, x_k$	2	Ц	Д
	AMIN1 ($X_1,\ldots,X_k$)		2	Д	Д
	MIN0 ($X_1,\ldots,X_k$)		2	Ц	Ц
	MIN1 ($X_1,\ldots,X_k$)		2	Д	Ц
	DMIN1 ($X_1,\ldots,X_k$)		2	ДР	ДР
Абсолютная величина числа	IABS (X)	$\|x\|$	1	Ц	Ц
	ABS (X)	"	1	Д	Д
	DABS (X)	"	1	ДР	ДР
Преобразование целого числа в действительное и DOUBLE PRECISIOH	FLOAT (X)	Число в форме REAL и DOUBLE PRECISION	1	Ц	Д
	DFLOAT (X)		1	Ц	ДР

Содержание программы	Вид обращения	Значение функции	Количество аргументов	Тип аргумента	Тип функции
Преобразование действительного числа в целое	IFIX (X)	Целое	1	Д	Ц
Присвоение знака второго аргумента абсолютной величине первого аргумента	ISIGN (X, Y) SIGN (X, Y) DSIGN (X, Y)	x^* sign (y) " "	2 2 2	Ц Д ДР	Ц Д ДР
Преобразование числа из формы REAL в форму DOUBLE PRECISION	DBLE (X)	Число в форме DOUBLE PRECISION	1	Д	ДР
Преобразование числа из формы DOUBLE PRECISION в форму RFAL	SNGL (X)	Число в форме REAL (старшая часть аргумента)	1	ДР	Д

Примечание. Д означает число в форме REAL, Ц — число в форме INTEGER, ДР — число в форме DOUBLE PRECISION.

ся преобразованию, может иметь любое представление, допускающее это преобразование.

Подбиблиотека функций, предназначенных для обработки строк, позволяет пользователю выделять из строк части (подстроки), находить расположение в строке заданной последовательности символов или битов, встраивать в заданное место строки другую строку и т. п. В этой подбиблиотеке имеются средства, облегчающие построение сложных строк из простых, позволяющие очищать память или заполнять ее определенными комбинациями символов, находить внутреннее битовое представление аргумента.

Аргументы очень многих функций этих трех подбиблиотек могут быть не только скалярными переменными, но и массивами. В последнем случае вырабатываемый функцией результат будет массивом той же размерно-

Таблица 3.14

Содержание программы	Вид обращения	Значение функции	Количество аргументов	Характеристика аргумента	Характеристика функции
Показательная функция	EXP (X)	e^x	1	Арифметические	
Натуральный логарифм	LOG (X)	$\ln x$	1	»	
Десятичный логарифм	LOG10 (X)	$\log_{10} x$	1	»	
Двоичный логарифм	LOG2 (X)	$\log_2 x$	1	»	
Синус	SIN (X)	$\sin x$	1	В радианах	Арифметическая
	SIND (X)	»	1	В градусах	То же
Косинус	COS (X)	$\cos x$	1	В радианах	»
	COSD (X)	»		В градусах	»
Тангенс	TAN (X)	$\mathrm{tg}\, x$	1	В радианах	»
	TAND (X)	»	1	В градусах	»
Арктангенс	ATAN (Y, X)	$\mathrm{arctg}\, x$ или $\mathrm{arctg}\,(y/x)$	1 или 2	Арифметический	В радианах
	ATAND (Y, X)		1 или 2	»	В градусах
Гиперболические:					
синус	SINH (X)	$\mathrm{sh}\, x$	1	Арифметические	
косинус	COSH (X)	$\mathrm{ch}\, x$	1	»	
тангенс	TANH (X)	$\mathrm{th}\, x$	1	»	
арктангенс	ATANH (X)	$\mathrm{arth}\, x$	1	»	
Квадратный корень	SQRT (X)	$\sqrt{x}$	1	Положительный	Арифметическая
Интеграл вероятностей	ERF (X)	$\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$	1	Арифметические	
Целая часть числа	FLOOR (X)	$[x]$	1	Арифметический	Целая
Абсолютная величина числа	ABS (X)	$\lvert x \rvert$	1	Арифметические	

Знак величины	SIGN (X)	1 для $x > 0$ 0 для $x = 0$ —1 для $x < 0$	1	Арифметический	Целая
Округление до ближайшего большего целого	CELL (X)	$-[-x]$	1	»	»
Округление до ближайшего меньшего по абсолютной величине целого	TRUNC (X)	$[\|x\|]\ \mathrm{sign}(x)$	1	»	»
Округление до заданного числа знаков	ROUND (X, n)	Округленное значение x	2	Арифметические	
Максимальное значеие	MAX $(X_1,\ldots,X_k)$	$\max\{x_i\}$ $i=1, 2,\ldots, k$	$\geqslant 2$	»	
Минимальное значение	MIN $(X_1,\ldots,X_k)$	$\min\{x_i\}$ $i=1, 2,\ldots, k$	$\geqslant 2$	»	
Преобразование переменной к заданной разрядности	PRECISION (X, P [,Q])	Переменная с заданной разрядностью	2 или 3	X — арифметический, P, Q — целые	Арифметическая
Преобразование любой переменной в переменную с плавающей точкой	FLOAT (X [,P])	Переменная X преобразуется в переменную с плавающей точкой разрядности P	1 или 2	X — любой P — целый	С плавающей точкой
Преобразование любой переменной в переменную с фиксированной точкой	FIXED (X, [P [,Q]])	Переменная X преобразуется в переменную с фиксированной точкой	От 1 до 3	X — любое, P, Q — целые	С фиксированной точкой
Преобразование любой переменной в арифметическую переменную с описателем DECIMAL	DECIMAL (X [,P [,Q]])	Переменная X преобразуется в арифметическую переменную с описателем DECIMAL	От 1 до 3	X — любое, P, Q — целые	С описателем DECIMAL
Преобразование любой переменной в арифметическую переменную с описателем BINARY	BINARY (X [,P [,Q]])	Переменная X преобразуется в арифметическую переменную с описателем BINARY	От 1 до 3	X — любое, P, Q — целые	С описателем BINARY

Продолжение табл. 3.14

Содержание программы	Вид обращения	Значение функции	Количество аргументов	Характеристика	
				аргумента	функции
Преобразование любой переменной в битовую строку	BIT (X [,P])	Переменная X преобразуется в битовую строку длины P	1 или 2	X — любой, P — целый	Битовая строка
Преобразование любой переменной в символьную строку	CHAR (X [,P])	Переменная X преобразуется в символьную строку длины P	1 или 2	То же	Символьная строка
Построение строки 16-ричных FF	HIGH (l)	Строка FF длины *l*	1	Целый	То же
Построение строки 16-ричных нулей	LOW (l)	Строка 00 длины *l*	1	»	»
Выделение подстроки из символьной или битовой строки (может использоваться для вставки в строку другой строки или ее части)	SUBSTR (X, P, l)	Строка знаков или битов	3	X — строка, P — выражение, l — целое	Строка того же типа, что и X
Определение позиции, начиная с которой строка Y впервые встречается в качестве подстроки в строке X)	INDEX (X, Y)	Целое двоичное число	2	Строки символов или битов	Целое число
Определение внутреннего представления переменной в виде битовой стро-	UNSPEC (X)	Битовая строка	1	Любая переменная длиной до 8 байтов	Битовая строка

кн (может использоваться для присвоения аргументу иекоторого внутреннего представления)					
Построение строки путем многократного сцепления другой строки	REPEAT (X, M)	Строка, получаемая в результате сцепления	2	X — строка, M — целое	Строка того же типа, что и X
Логическая сумма элементов массива	ANY (X)	Битовая строка, получаемая логическим суммированием внутреиних представлений элементов массива	1	Массив	Битовая строка
Логическое произведение элементов массива	ALL (X)	Битовая строка, получаемая логическим перемножением внутренних представлений элементов массива	1	„	То же
Арифметическая сумма элементов массива	SUM (X)	Сумма элементов массива	1	Массив арифметического типа	Число с плавающей точкой
Арифметическое произведение элементов массива	PROD (X)	Произведение элементов массива	1	То же	То же
Определение адреса переменной	ADDR (X)	Адрес переменной	1	Имя переменной	Целое число
Определение текущей даты	DATE	Текущая дата в виде строки „ГГММДД"	—	—	Строка знаков
Определение текущего времени	TIME	Текущее время в виде строки „ЧЧММССТТТ"	—	—	То же

Примечание. Число Q — количество знаков после десятичной точки (указывается только для переменных с фиксированной точкой); ГГ — последние две цифры года, ММ — порядковый номер месяца, ДД — число месяца, ЧЧ — часы, ММ — минуты, СС — секунды, ТТТ — тысячные доли секунды.

сти. Но в библиотеке ПЛ/1 имеется подбиблиотека функций, оперирующая специально с массивами. Содержащиеся в ней функции позволяют находить логическую сумму или логическое произведение элементов битовых массивов, произведение или сумму элементов массивов арифметического типа.

Последняя, пятая подбиблиотека содержит функции, позволяющие находить дату и время суток, адреса переменных и некоторые другие данные.

Перечень и характеристики функций библиотеки ПЛ/1 ДОС/ЕС приводятся в табл. 3.14.

4. Привязка языков программирования к ДОС/ЕС

4.1. Описание файлов

Основным отличием языков программирования, таких как ФОРТРАН, ПЛ/1, КОБОЛ, от языков описания алгоритмов или алгоритмических языков (таких, например, как АЛГОЛ-60) является возможность средствами языка обращаться к наборам данных (файлам), размещенным на внешних носителях, и в первую очередь осуществлять средствами языка разнообразные операции ввода — вывода данных. Для этого в языке должны иметься средства описания файлов.

В этом отношении Базисный ФОРТРАН предоставляет программисту сравнительно малые возможности. Описание осуществляется с помощью оператора DEFINE FILE, помещаемого в описательной части программы (одним таким оператором можно описать несколько файлов). Вид этого оператора

DEFINE FILE <список>

где элементы списка, разделяемые запятыми, имеют вид

<имя файла>(<число записей>, <максимальный размер записи>, <параметр формата и структуры файла>, <текущий номер записи>).

Параметр формата и структуры файла может принимать значения: L, если максимальный размер записи задается в байтах; E, если максимальный размер задается также в байтах, но передача осуществляется с исполь-

зованием формата; U, если максимальный размер задается в машинных словах, а передача осуществляется без использования формата.

Текущий номер записи обозначается простой неиндексированной переменной.

Именем файла в Базисном ФОРТРАНе может быть только целое число без знака.

Например,

DEFINE FILE 6(100, 300, E, M)

означает определение на SYS003 файла из 100 записей не более чем по 300 байтов каждая, передаваемых с ис-

Таблица 4.1

Номер файла (имя файла в программе)	Имя логического устройства	Имя файла в ДОС/ЕС	Допустимый тип операции	Допустимый тип устройства
1	SYSIPT	IJSYSIN	Ввод	Считыватель перфокарт, магнитная лента, диск
1	SYSIN	IJSYSIN	Ввод	То же
2	SYSPCH	IJSYSPH	Вывод	Карточный перфоратор, магнитная лента, диск
3	SYSLST	IJSYSLS	Вывод	АЦПУ, магнитная лента, диск
4	SYS001	IJSYS01	Ввод или вывод	Считыватель перфокарт, карточный перфоратор, АЦПУ, магнитная лента, диск
5	SYS002	IJSYS02		
6	SYS003	IJSYS03		
7	SYS004	IJSYS04		
8	SYS005	IJSYS05		
9	SYS006	IJSYS06		
10	SYS007	IJSYS07		
11	SYS008	IJSYS08		
12	SYS009	IJSYS09		
13	SYS010	IJSYS10		
14	SYS011	IJSYS11		
15	SYSLOG	—	Вывод	Пишущая машинка, АЦПУ

Примечание. Обозначение SYSIN используется при назначении для SYSRDR и SYSIPT одного и того же физического устройства.

пользованием формата, а M — текущий номер записи. Транслятор с Базисного ФОРТРАНа позволяет оперировать с пятнадцатью файлами, описываемыми в табл. 4.1. Четыре из них закреплены за системными логическими устройствами ввода — вывода, а остальные — за устрой-

ствами программиста. При этом запрещается использовать многотомные файлы, файлы на дисках, состоящие из многих участков (этим исключается возможность применения индексно-последовательной организации файлов), считывать файлы с многофайлового тома на магнитной ленте, а также записывать несколько файлов на одну магнитную ленту.

В ПЛ/1 таких ограничений нет, и для описания файлов программисту предоставляются значительно бо́льшие возможности. Среди описателей оператора DECLARE для файлов имеется описатель FILE, который должен следовать первым, непосредственно за именем файла. Этот описатель вместе с остальными составляет список описателей файла, имеющий вид

<список описателей файла> : : =FILE <способ передачи данных><характеристика запоминающей среды><область действия>

Здесь:

<способ передачи данных> : : =RECORD <описатели файла при передаче записями>|[STREAM] <описатели файла при передаче потоком>

Описатель STREAM можно не указывать, так как отсутствие описателя RECORD рассматривается транслятором как наличие описателя STREAM.

Описатели файлов, зависящие от способа передачи, описываются правилами:

<описатели файла при передаче потоком> : : = <назначение файла>

<описатели файла при передаче записями> : : = <назначение файла><способ буферизации>|<способ доступа><другие описатели>

где

<способ буферизации> : : = [BUFFERED| UNBUFFERED]

<способ доступа> : : = [SEQUENTIAL|DIRECT]

<другие описатели> : : = [<наличие ключа>] [<допустимость чтения при обратном движении>]

При передаче потоком:

<назначение файла> : : = INPUT|OUTPUT|PRINT [OUTPUT]

а при передаче записями:

<назначение файла> : : = INPUT|OUTPUT| UPDATE

Для характеристики запоминающей среды применяется описатель

ENVIRONMENT(<список режимов ввода—вывода>)

Список, в свою очередь, строится по правилам:

<список режимов ввода — вывода> : : = <режим ввода — вывода>|<список режимов ввода — вывода> <режим ввода — вывода>

<режим ввода—вывода> : : = <устройство>|<тип записей>|<способ организации>

Отдельные режимы описываются правилами

<устройство> : : = MEDIUM (SYSxxx, nnnn)

где SYSxxx — имя логического УВВ, например SYSLST или SYS007, а nnnn — шифр физического устройства: 6012 — для ввода перфокарт, 5052 или 5056 — для накопителей на дисках и т. п. Из системных устройств разрешается использование SYSLST, SYSIPT и SYSPCH, а выбор логических устройств программиста не ограничивается.

При указанни типа записей учитывается их блочность, если она применяется. В соответствии с этим

<тип записей> : : = F(<длина блока>, <длина записи>)|F(<длина записи>)|V(<длина блока>)|U (<длина записи>)

Первая форма F употребляется для блокированных записей фиксированной длины, а вторая для неблокированных.

Способ организации файла задается в соответствии с возможностями ДОС/ЕС:

<способ организации> : : = <последовательная организация>|<прямая организация>|<индексно-последовательная организация>

Для каждой из них имеется свой набор описателей:

<последовательная организация> : : = CONSECUTIVE [<буферизация>][<управляющие символы>]<режимы ленточных файлов>

<прямая организация> : : = {REGIONAL(1)|

REGIONAL(3)} [<количество участков>] [<режимы дисковых файлов>]

<индексно-последовательная организация> : : = INDEXED <режимы дисковых файлов><режимы индексно-последовательных файлов>

Для дополнительных описателей

<буферизация> : : = BUFFERS(1) | BUFFERS(2)

причем по умолчанию осуществляется режим BUFFERS(1), а для файлов с описателем UNBUFFERED режим BUFFERS запрещен;

<управляющие символы> : : = STLASA | STLYES

и применяются они для файлов RECORD OUTPUT на логических устройствах программиста, устройствах системной печати и перфорации;

<режимы ленточных файлов> : : = [LEAVE]

[NOLABEL] [NOTAPEMK]

позволяют программисту выполнять операции, эквивалентные действию оператора // МТС УПРАВЛЕНИЯ ЗАДАНИЯМИ (см. п. 7.2.1).

Для прямой организации

<количество участков> : : = EXTENTNUMBER(n)

позволяет программисту строить файлы из произвольного числа несмежных участков;

<режимы дисковых файлов> : : = <режим проверки> <количество участков><длина ключа>

причем операнд <длина ключа> не используется в файлах REGIONAL (1), а для других указывается в виде

KEYLENGTH (n)

Режим проверки VERIFY применяется только для файлов на дисках с описателями OUTPUT или UPDATE.

Режимы индексно-последовательных файлов позволяют:

— построить главный индекс, указав режим

INDEXMULTFPLE

— определить число дорожек переполнения на каждом цилиндре с помощью режима

OFLTRACKS (n)

— указать положение первой позиции ключа в записи

KEYLOC (n)

Имеются и другие режимы для этой категории файлов.

Пусть, например, требуется описать файл прямого доступа на логическом устройстве SYS015, которому выделено физическое устройство X'190'. Имя файла ABC123, записи блокированные, длина записи 60 байтов, длина блока 600 байтов, в записях имеется поле ключа длиной 5 байтов, первая позиция которого находится в десятой позиции записи. Файл предполагается использовать в режиме обновления.

Решение.

DECLARE ABC123 FILE RECORD UPDATE DIRECT KEYED ENVIRONMENT (MEDIUM (SYS015, 5056) F(600, 60) REGIODAL (3) KEYLOC (10) KEYLGTH (5));

Здесь описатель KEYED показывает наличие в записях поля ключа, а описатели KEYLOC и KEYLGTH определяют его расположение и длину.

Возможность чтения ленточного файла в обратном направлении задается режимом

BACKWARDS

Все режимы записываются друг за другом и отделяются по меньшей мере одним пробелом.

Среди описателей файла имеется еще описатель области действия

EXTERNAL

который, однако, может явно не указываться, так как выбирается системой и по принципу умолчания.

4.2. Операторы ввода—вывода

Операторы ввода — вывода в Базисном ФОРТРАНе ДОС/ЕС позволяют производить передачу данных как с преобразованием их формата, так и без преобразования. Эти операторы бывают двух видов — для работы

с файлами последовательного и прямого доступа. Для преобразования формата применяется оператор FORMAT, работающий на правах описания (он может располагаться в любом месте программной единицы).

Работа с файлами последовательного доступа ведется с помощью операторов ввода — вывода с ключевыми словами READ, WRITE, REWIND, ENDFILE и BACKSPACE, а с файлами прямого доступа — с помощью операторов с ключевыми словами READ, WRITE и FIND.

ПЛ/1 для операций ввода — вывода располагает операторами, работающими ближе к физическому уровню, в так называемом режиме *передачи записями*,— это оператор-READ и оператор-WRITE, и операторами, работающими на более высоком уровне — *передаче потоком* — это оператор-GET и оператор PUT. Передача записями возлагает на программиста больше обязанностей, но работает она несколько быстрее. При передаче потоком значительная часть работы программиста перекладывается на систему и за счет этого передача идет медленнее. Таким образом, при передаче записями как бы в меньшей степени используются возможности алгоритмического языка, чем это позволяет передача потоком. Но при передаче потоком возможности программиста сужаются, и это заставляет его иногда переходить на более низкий уровень использования языковых средств.

Основными операторами в обоих языках являются операторы ввода данных с внешнего носителя в оперативную память машины и операторы вывода из оперативной памяти на внешний носитель. Внешним носителем в общем случае могут быть не только перфокарты и бумага АЦПУ, но и магнитные ленты и диски.

4.2.1. Операторы ввода. В Базисном ФОРТРАНе оператором ввода является оператор READ, существующий в четырех вариантах:

— оператор-READ последовательного доступа форматизованный и неформатизованный;

— оператор-READ прямого доступа форматизованный и неформатизованный.

Общий вид оператора

READ(<номера файла>[<'номер записи>][,<метка оператора FORMAT>])<список ввода>

Номер файла — целая константа без знака или целая переменная, принимающая положительные значения. Номер записи — целое выражение, отделяемое от номера файла апострофом. Для оператора последовательного доступа апостроф и номер записи пропускаются. Метка оператора FORMAT указывает место в программе оператора, определяющего способы преобразования и порядок размещения вводимых данных. Для неформатизованных вариантов оператора ввода эта метка и предшествующая ей запятая пропускаются.

Список ввода в тривиальном случае может отсутствовать, и тогда оператор не выполняется. Общий вид списка ввода

<список ввода> : : = <элемент списка> | <список ввода> , <элемент списка>

где

<элемент списка> : : = <имя переменной> | <имя массива> | <неявный оператор DO> .

Имя переменной может обозначать простую или индексированную переменную. Имя массива указывается в случае, когда необходимо переслать его полностью. При необходимости же переслать часть массива, например строку или столбец матрицы, употребляется так называемый неявный оператор DO: пишется имя массива с указанием индексов, часть которых может быть закреплена, а остальные подлежат перебору. Далее после запятой указывается имя перебираемого индекса и после знака равенства его начальное и конечное значения, а если шаг изменения индекса отличен от единицы, то и значение шага. Возможно использование вложенных неявных операторов по аналогии с вложенными циклами.

Пример 16. Из файла 8 последовательного доступа ввести в память переменную A, элемент массива B (4) и строку C (3,1), C(3.2),...,C(3,M) матрицы C, адрес оператора FORMAT 21.

Вид оператора ввода:

```
READ (8,21) A,B(4),(C(3,K), K=1,M)
```

Пример 17. Из файла 12 примого доступа ввести 10 записей с разиесением их содержания согласио списку: A(I),I=1,2,...,M, B(4,K), где K — номер записи; C(I,J),I=1,2,...,M, J=1,2,... Оператор формата снабжен меткой 6. Отрезок программы будет иметь вид:

```
  D04K=1,10
4 READ(12'K,6)(A(I),I=1,M),B(4,K),((C(I,J),I=1,M),J=1,N)
```

Состав преобразований, выполняемых в ФОРТРАНе при вводе и выводе, описываются в операторе FORMAT:

<метка> FORMAT(<список форматов>)

Метка в этом операторе обязательна. Список форматов, как обычно, состоит из элементов списка, разделенных запятыми, причем

<элемент списка формата> : : = [<кратность>] {<код преобразования><число позиций>.<длина дробной части>|<число позиций><код преобразования>|<литерал>}

Кратность элемента списка указывает число его повторений, причем возможно отнесение кратности к нескольким элементам, и тогда она пишется впереди такой группы, заключенной в скобки. Обозначается кратность целой константой без знака, не превышающей 256. В качестве кодов преобразования употребляются прописные латинские буквы I, E, F, D, A, H, X и T.

Код I применяется или для передачи целых констант, или для передачи данных, которые должны быть преобразованы к целому виду. Число выделяемых при передаче позиций ставится вслед за кодом без пробела. Например, I12 означает, что из 12 позиций перфокарт или 12 байтов другого носителя должны быть введены цифры целого числа; 4(I12, I6, I9) означает, что во вводимом (а для оператора вывода выводимом) потоке данных четыре раза выделяются группы, в каждой из которых первое поле занимает 12-разрядное целое число, за ним следует 6- и 9-разрядное.

Код F применяется для ввода и вывода чисел с фиксированной точкой (т. е. без указания степени десятки, на которую должно быть умножено число). Длина дробной части указывается, если это необходимо, и в этом случае при определении числа позиций должно быть предусмотрено место для десятичной точки.

Например, F6,3F8,2(F10,3,F12,5) означает выделение одного шестизначного и трех восьмизначных полей, а за ними двух групп, в каждой из которых первое поле состоит из 10 позиций, в том числе 3 после десятичной точки, а второе из 12 позиций, из которых 5 после десятичной точки.

Код E нужен в ФОРТРАНе для ввода — вывода чисел, представленных в форме с плавающей точкой, т. е. с указанием степени числа 10, на которую умножается число. Для такого числа необходимо выделить количество позиций для размещения его знака, десятичной точки (если необходимо), символов E или D, знака и цифр порядка. Так, для числа — 3141.592E—03 должно быть выделено не менее 13 позиций.

Для чисел с двойным количеством разрядов вместо кода E употребляется код D.

Если ввести с перфокарт два целых восьмизначных числа, число с фиксированной точкой, размещаемое не более чем в десяти позициях с двумя знаками в дробной части, а также два числа с плавающей точкой, требующих по 16 позиций, в том числе 4 для дробной части, то оператор-FORMAT нужно записать в виде

<метка> FORMAT(2I8,F10.2,2D16.4)

Для кода F при вводе и выводе, а для кодов E и D только при выводе может применяться масштабный множитель вида

<порядок>P

размещаемый непосредственно перед кодом или перед кратностью, если она есть. Например, 2PF12.5 означает, что у числа, имеющего 5 знаков в дробной части, десятичная точка должна быть сдвинута на две позиции вправо.

Код A применяется для ввода и вывода символьных констант, не предназначенных для арифметических действий над ними. Число позиций указывается, как и для числовых констант, справа от кода. При вводе данных с кодом A символьным материалом разрешается заполнять также поля, выделенные для числовых констант.

Для передачи символьных данных в ФОРТРАНе существуют и другие средства: код H и передача литералов.

Код H вызывает при выводе передачу того текста, который расположен внутри списка форматов, причем число выделяемых позиций в этом случае необходимо располагать слева от кода и рассматривать как кратность передачи символьных строк длиной в 1 символ. При вводе код H позволяет иметь в программе оператор-FORMAT с пустым списком форматов и заполнять его через вводной файл.

Аналогично работает элемент списка форматов литерального типа.

Так, операторы

FORMAT(19H ВВЕДИТЕ ЭТОТ ТЕКСТ)
FOFMAT ('ВВЕДИТЕ ЭТОТ ТЕКСТ')

вызовут одинаковый эффект.

С помощью кода X можно при вводе пропускать заданное число позиций.

Код T позволяет строить всевозможные комбинации из буквенных текстов и значений вычисляемых переменных.

В ПЛ/1 имеются два оператора ввода: оператор ввода при передаче записями, использующий как и в ФОРТРАНе, ключевое слово READ, и оператор ввода при передаче потоком с ключевым словом GET.

При передаче потоком предполагается, что передаваемая информация состоит из однородного потока символов, которые при передаче преобразовываются в соответствии с имеющимся в операторе ввода (или вывода) списком элементов формата (аналогичных употребляющимся в операторе FORMAT языка ФОРТРАН). Записи во внешней памяти при передаче потоком блокировать не разрешается. Доступ к записям обязательно последовательный.

При передаче записями, наоборот, допускается использование всех способов организации файлов, разрешается блокирование записей, но нельзя попутно с передачей преобразовывать данные.

Вид оператора ввода при передаче записями

READ <список режимов>

причем первым в списке режимов должен стоять режим

FIL(<имя файла>)

а остальные следовать за ним через пробелы.

Имя файла, имеющее по умолчанию описатель EXTERNAL, должно быть объявлено в операторе DECLARE и иметь описатели направления передачи INPUT или UPDATE. До появления в программе оператора-READ файл с этим именем должен быть явно открыт с помощью оператора-OPEN.

Имя переменной, в область которой пересылается вводимый материал, указывается в режиме INTO:

INTO(<идентификатор>)

оно должно быть именем простой переменной, массива или структуры. Использование здесь имени индексированной переменной или имени объекта, входящего в структуру, не разрешается.

Для файлов с описателями DIRECT и SEQUENTIAL INDEXED (и только для них) задается режим

KEY(<выражение>)

в котором <выражение> должно быть такого вида, чтобы его можно было преобразовать в строку знаков, используемую в качестве ключа.

При описателе DIRECT и файлах с описателем REGIONAL(1) эта строка состоит только из десятичных цифр и имеет длину 8 байтов. При описателе REGIONAL(3) ключ может иметь длину от 9 до 255 байтов, из которых последние 8 — десятичные цифры. Длина ключа у всех записей одного файла должна быть одинаковой.

Для файлов с описателями SEQUENTIAL INDEXED режим KEY можно не задавать, но в этом случае ввод файла оказывается возможным только в последовательном режиме. Но для таких файлов имеется еще режим KEYTO, который позволяет переменной в скобках

KEYTO(<имя>)

присваивать имеющееся в записи значение ключа.

Режимы KEY и KEYTO взаимно исключают друг друга.

Режим SET, применяемый при использовании базированных переменных, здесь не рассматривается.

При передаче потоком оператор ввода имеет вид

GET [<режим передачи>]EDIT<набор списков>

где

<режим передачи> : : = FILE(<имя файла>) | STRING(<имя>) | <пусто>

а набор списков, разделяемых пробелами, состоит из пар

(<список ввода>)(<список элементов формата>)

Оператор ввода позволяет вводить данные из внешней памяти в области, отведенные для переменных, или перемещать их из одного участка оперативной памяти в другой. При перемещении производится преобразование согласно списку форматов.

Отсутствие режима передачи означает, что ввод производится из стандартного файла системы.

Режим STRING в операторе GET позволяет при передаче данных производить преобразования, которые выполнять другими средствами запрещено, например, из строчных в арифметические.

Список ввода и список элементов формата должны состоять из попарно соответствующих друг другу элементов: в первом из них перечисляются имена вводимых данных, а во втором — их форматы.

Имена вводимых данных в списке ввода отделяются друг от друга запятыми. Вводиться могут простые или индексированные переменные, массивы, структуры или любые входящие в них объекты. В качестве имени вводимых данных может использоваться также псевдопеременная, т. е. имя библиотечной функции, находящееся в левой части оператора присваивания. Кроме имен, по аналогии с ФОРТРАНом в списке ввода может быть некоторое подобие группы — список ввода со спецификацией DO, например:

(X(J) DO J=1 TO 12)

или

(((ABC12(K, M) DO K=1 TO 20) DO M=1 TO 100))

Таким образом, оператор

GET EDIT (A.B.C,E,X,((H(K,M) DO K=1 TO 5) DO M=1 TO 10)) (<список форматов>)

означает ввод элемента структуры A.B.C, простой переменной E, массива X (предполагается, что X описан как массив) и матрицы H(K, M) с 10 строками и 5 столбцами.

Элементы формата при вводе — это или элемент формата данных или элемент формата R или же, наконец, элемент формата X. Как и в ФОРТРАНе, перед элементом формата может стоять его кратность. Эле-

мент формата R является аналогом оператора FORMAT ФОРТРАНа. Он позволяет описывать форматы не внутри операторов ввода — вывода, а в другом месте программы.

Элемент формата данных при вводе числа с фиксированной точкой имеет вид

F(<число позиций>[,<длина дробной части>
[,<масштабный множитель>]])

Число позиций указывает количество колонок на перфокарте или байтов на другом носителе, откуда считывается информация о данном объекте, и может изменяться от 1 до 32. Длина дробной части указывается в случае необходимости, определяя положение десятичной точки в численном представлении переменной. Масштабный множитель (целое число со знаком) позволяет изменить положение десятичной точки, извлекаемой вместе с числом из входного потока. Если масштабный множитель отсутствует, то при наличии десятичной точки во входном потоке данные о ее расположении в списке ввода игнорируются, а при отсутствии берутся из этого списка.

Элемент формата данных при вводе числа с плавающей точкой имеет вид

E(<число позиций>,<длина дробной части>
[,<число значащих цифр мантиссы>])

Число позиций и длина дробной части числа с плавающей точкой имеют тот же смысл, что и для формата F, число же значащих цифр используется лишь при выводе. Длину дробной части необходимо указывать и при ее отсутствии, и когда точка закодирована в представлении числа во входном потоке.

Для строчных переменных символьного типа имеется формат вида

A[(<число позиций>)]

причем число позиций должно находиться в пределах от 1 до 255.

Для строчных переменных битового типа применяется формат

B[(<число позиций>)]

но число позиций в этом случае может лежать только в пределах от 1 до 64.

Элемент формата X, аналогично такому же элементу в ФОРТРАНе, позволяет пропускать указанное в нем число позиций. Его вид

X(<число позиций>)

Последним мы рассмотрим элемент формата R вида

R(<имя>)

где в качестве имени указывается метка оператора FORMAT, который, в свою очередь, может содержать список всех перечисленных элементов формата, кроме R.

Приведем несколько примеров.

Пример 18. Пусть требуется ввести с перфокарт строчиую переменную A, состоящую из 40 символов, 5 чисел, у первого и четвертого из которых дробной части нет, а остальиые имеют по два дробных разряда, причем эти числа входят в структуру B в качестве скаляриых переменных C1,C2,C3 и массива C из двух элементов. В структуру входит еще символьная переменная K длиной 10 байтов. Во избежание ошибок смещения символьные переменные помещены на первой перфокарте подряд в 50 первых позициях, а каждое число — иа отдельной перфокарте в последних 10 позициях.

Р е ш е н и е. Оператор ввода для такого размещения даиных можно записать так:

```
GET EDIT(A,B.K,B.C1,B.C2,B.C3.B.C)(A(40),A(10),X(30),
R(M12));
M12:FORMAT(5(X(70),F(10)));
```

Пример 19. Ввести структуру, в состав которой входят M числовых даиных и расположениые вслед за ними их размерности. Числовые даниые занимают на перфокарте по 10 позиций (в том числе 4 после десятичиой точки), размерности — по 6 символов. Остальные позиции перфокарты пустые. При вводе размерности игиорировать.

Р е ш е и и е. Чтобы элемент типа X не оказался последним, а он в этой позиции. как известио, не выполняется (это может привести к сдвигу вводимых данных и появлению семантических ошибок), введем фиктивиую символьную переменную AAA. После этого оператор ввода можно записать так:

```
GET EDIT(((X(K),Y(K),AAA) DO K=1 TO M))(F(10,4),A(6),
A(70));
```

4.2.2. Операторы вывода. Аналогично оператору ввода в Базисном ФОРТРАНе имеется четыре варианта оператора вывода:

— оператор-WRITE последовательного доступа форматизованный и неформатизованный;

— оператор-WRITE прямого доступа форматизованный и неформатизованный;

— оператор-WRITE последовательного доступа неформатизованный;

— оператор-WRITE прямого доступа неформатизованный.

Вид этого оператора

WRITE(<номер файла>['<номер записи>][,<метка оператора FORMAT>])<список вывода>

Способ кодирования и разделения номеров файла и записи, а также метки оператора FORMAT в операторе WRITE аналогичны описанным для оператора-READ.

Косая черта, разделяющая части списка оператора FORMAT, относящиеся к разным записям, позволяет управлять пропуском строк. Так, например, оператор

<метка> FORMAT(8E10.3///8E10.3)

вызывает печать восьми чисел формата E с тремя знаками после десятичной точки, а потом, после пропуска трех строк, печать еще одной такой же строки.

При выводе данных на печать в операторе FORMAT должен быть управляющий символ, обеспечивающий возврат каретки к началу строки. Это или 'X' или же 1Hx, где x может быть пробелом или одним из трех знаков: 0, 1 или +;

1H вызывает продвижение каретки перед началом печати на одну строку;

1H0 переводит каретку на две строки;

1H1 вызывает начало печати на первой строке следующей страницы;

1H+ приводит к тому, что печать начинается без продвижения строки.

Символы управления движением каретки на АЦПУ не печатаются. При передаче же данных на устройство, назначенное для SYSLST вместо АЦПУ, например накопитель на магнитной ленте, они сохраняются и выполняют свою роль при перемещении данных с этого устройства на реальное устройство печати.

Для формирования заголовков различных таблиц в ФОРТРАНе применяется код T, позволяющий разносить данные в определенные позиции строки. С помощью этого кода можно строить многострочные заголовки.

Пример 20. Построить оператор вывода для заголовка табл. 4.1.

Р е ш е н и е. Операторы печати и формата можно написать в виде

```
WRITE(3,5)
5 FORMAT (T2,'НОМЕР', T11, 'ИМЯ ЛОГИЧЕС-', T23, 'ИМЯ ФАЙ-
ЛА В', T37, 'ДОПУСТИМЫЙ', T51, 'ДОПУСТИМЫЙ'/T2,'ФАЙЛА',
T11, 'КОГО УСТРОЙ-', T23,'ДОС/ЕС', T37, 'ТИП ОПЕРА-', T51,
```

Этот код в сочетании с другими (а не только с литеральными, как показано в примере) позволяет строить всевозможные комбинации из буквенных текстов и значений вычисляемых переменных.

Оператор WRITE в ПЛ/1, употребляемый при передаче данных записями, может применяться не только для вывода данных во внешнюю память, но и для их обновления (передаются как блокированные, так и неблокированные записи). Оператор WRITE по аналогии с оператором READ имеет вид

WRITE <список режимов>

и первым режимом должен быть

FILE(<имя файла>)

Остальные режимы должны следовать за этим через пробелы в произвольном порядке.

Отсутствие режима FILE означает, что обмен происходит со стандартным файлом системы. В остальных случаях файл должен быть описан в операторе DECLARE с описателями RECORD и OUTPUT или UPDATE и явно открыт с помощью оператора OPEN.

Роль режима INTO в операторе WRITE играет режим FROM

FROM(<идентификатор>)

Здесь идентификатор не может быть именем какого-либо объекта, входящего в более крупный объект, т. е. индексированной переменной или элементом структуры.

Кроме этих обязательных режимов оператора WRITE используется еще режим

KEYFROM(<выражение>)

с выражением скалярного типа, допускающим преобразование к виду строки знаков. Этот режим применяется для файлов с описателями DIRECT или SEQUENTIAL INDEXED. Он допускает замену записи в файлах REGIONAL (1), если выражение в режиме KEYFROM является ключом этой записи. Для файлов REGIONAL

(3), где запись делится на область ключа и область данных, выражение при KEYFROM заносится в область ключа. В область данных же заносится информация, определяемая идентификатором режима FROM. Если несколько выражений определяет одно и то же значение ключа, то более поздняя информация уничтожает предыдущую.

Для индексно-последовательных файлов с блокированными записями в область ключа блока заносится ключ последней входящей в этот блок записи. Положение ключа в записи и его длина определяются описателями KEYLOC и KEYLENGTH оператора DECLARE, в котором описан файл.

Пример 21. Файл ABCDE индексно-последовательной организации необходимо записать на магнитном диске в порядке возрастания ключей. Файл состоит из записей фиксированной длины по 320 байтов, из которых для ключа выделено 10 байтов, начиная с 50-го. Информация, подлежащая передаче, находится в массиве X124, а ключом является строчная переменная A22.

Решение. Оператор DECLARE для файла можно записать в виде

```
DECLARE ABCDE FILE RECORD SEQUENTIAL KEYED OUT-
PUT ENVIRONMENT (F(320) MEDIUM(SYS018,5052) INDEXED
EXTENTNUMBER(2) KEYLENGTH(10) KEYLOC(50));
```

Оператор записи имеет вид

```
WRITE FILE (ABCDE) FROM (X124) KEYFROM (A22);
```

Оператор вывода при передаче потоком имеет вид

PUT [<режим передачи>]EDIT<набор списков>

и опять

<режим передачи> : : =FILE(<имя файла>) | STRING(<имя>) | <пусто>

причем случай <режим передачи> : : = <пусто> означает использование стандартного файла системы. Набор списков в случае вывода имеет прежний вид: списки, разделяемые пробелами, состоят из пар

(<список вывода>) (<список элементов формата>)

Режим STRING в операторе PUT, как и в операторе GET, используется для перемещения данных в пределах оперативной памяти, но для случаев, когда требуется преобразование данных обратного смысла (например, из арифметических в строчные).

Кроме элементов формата, используемых при вводе, оператор PUT имеет форматы печати (задаваемые только для файлов с описателем PRINT).

Печать с новой страницы обеспечивается элементом формата PAGE, помещаемым среди элементов формата в том месте, где необходимо перейти к печати с новой страницы.

Печать с новой строки достигается включением в список элементов формата элемента SKIP, имеющего вид

SKIP(n)

где n — уменьшенное на единицу количество пропускаемых строк; если n не указано, то считается n=1; максимальное значение n=3.

Печать в заданной позиции обеспечивается элементом формата

COLUMN(n)

где $1 \leqslant n \leqslant 255$. Действие этого элемента подобно действию формата T в ФОРТРАНе. Если в указанную позицию строки еще ничего не занесено, то все позиции вплоть до нее заполняются пробелами, а если эта часть строки уже сформирована, то ее заполнение прекращается и начинается формирование следующей строки с указанной в элементе формата позиции.

Элемент формата LINE позволяет начать печать с заданной строки страницы. Его вид

LINE(n)

и, как и в предыдущих случаях, $1 \leqslant n \leqslant 255$.

Пример 22.

Пусть требуется оформить результаты лабораторных измерений в виде табл. 4.2:

Таблица 4.2

Результаты измерений, выполненных в лаборатории механического цеха 12 августа 1976 г.

Исполнитель А. В. Петров

Шифр металла	Номера партий						
	1	2	3	4	5	6	7
A1	31.12	30.94	31.21	30.88	30.79	32.01	31.16
B1	32.54	33.11	31.99	32.73	30.98	34.20	34.54

Измерения проводил

(Петров)

Решение. При выводе на стандартный файл печати имя файла указывать не требуется. Поэтому для ввода измеренных данных и формирования таблицы можно использовать программу

```
ABC: PROCEDURE OPTIONS (MAIN);
 DECLARE
 (X,Y)CHARACTER(5),(A1,A7)CHARACTER(20),
 A2 CHARACTER(63),A3 CHARACTER(22),A4 CHARACTER(13),
 A5 CHARACTER(4),(A6,A8)CHARACTER(8),B(7)DECIMAL
 FIXED, C(2,7)DECIMAL FIXED;
 DO K=1 TO 7; B(K)=K;END;
 A1='РЕЗУЛЬТАТЫ ИЗМЕРЕНИЙ';
 A2='ВЫПОЛНЕННЫХ В ЛАБОРАТОРИИ
 МЕХАНИЧЕСКОГО ЦЕХА'
 ||'12 АВГУСТА 1976 Г.';
 A3='ИСПОЛНИТЕЛЬ А.В.ПЕТРОВ';A4='НОМЕРА ПАРТИИ';
 A5='ШИФР'; A6='МЕТАЛЛА';A7='ИЗМЕРЕНИЯ
 ПРОВОДИЛ';
 A8='/ПЕТРОВ/';
GET EDIT(C) (F(6,2)); PUT EDIT (A1)
(PAGE, COLUMN (20),A);
 PUT EDIT(A2,A3) (SKIP,A,COLUMN(20),A);
 PUT EDIT((65)'—') (SKIP,A);
PUT EDIT(A5,A4) (SKIP,A,COUMN (25),A);
 PUT EDIT((57)'—') (COLUMN(8),A);
PUT EDIT(A6,B) (SKIP, A, 7 F(8));
 PUT EDIT((65)'—') (SKIP,A);
PUT EDIT('A1',(C(1,K)DO K=1 TO 7)) (SKIP,A,7F(8,2));
 PUT EDIT('B1', (C(2,K)DO K=1 TO 7)))SKIP,A,7F(8,2));
END ABC;
```

Приведенная здесь программа не может, конечно, служить образцом краткости. В частности, вместо шести операторов вывода можно было бы обойтись и одним, но тогда списки вывода и элементов формата оказались бы слишком длинными и ненаглядными. Кроме того, при разукрепнении списков ввода — вывода и форматов, а также операторов DECLARE облегчается отладка программы: диагностические сообщения транслятора указывают номер оператора, в котором содержится ошибка, но не всегда локализуют ее внутри оператора.

4.2.3. Вспомогательные операторы ввода — вывода. Кроме операторов READ и WRITE, Базисный ФОРТРАН располагает еще несколькими операторами, выполняющими вспомогательные функции: операторы REWIND, BACKSPACE и ENDFILE для последовательного доступа и оператор FIND для прямого доступа. Такие операторы, как DEFINE FILE и FORMAT, мы здесь не перечисляем, так как один из них был отнесен к описаниям, а другой по аналогии с ПЛ/1 был рассмотрен выше в качестве составной части операторов ввода и вывода.

Оператор-REWIND вида

REWIND a

где а — номер файла, заставляет очередной оператор ввода или вывода обращаться к первой записи файла. Он, таким образом, позволяет программисту изнутри программы на исходном языке выполнять функцию BSF оператора МТС УПРАВЛЕНИЯ ЗАДАНИЯМИ (см. ниже, п. 7.3.1.

Оператор BACKSPACE, имеющий аналогичный вид

BACKSPACE a

возвращает последующие операторы READ и WRITE назад на одну запись, подобно операнду BSP оператора МТС УПРАВЛЕНИЯ ЗАДАНИЯМИ.

Оператор ENDFILE вида

ENDFILE a

обеспечивает формирование конца файла аналогично операнду WTM оператора МТС.

Оператор прямого доступа FIND вида

FIND (a'r)

позволяет производить поиск записи с номером r в файле а в то время, когда программа еще обрабатывает предыдущую запись этого файла.

В языке ПЛ/1 при передаче данных записями имеется возможность заменить уже сделанную в файле запись. Для этого используется оператор-REWRITE вида

REWRITE <список режимов>

и первым среди режимов, как и в операторах READ и WRITE, указывается режим

FILE(<имя файла>)

но описание файла при этом должно содержать описатель UPDATE.

Кроме режима FILE, в операторе REWRITE предусмотрены режимы FROM и KEY (отсутствие режима FROM влечет за собой и отсутствие режима KEY).

Режим FROM имеет вид

FROM(<идентификатор>)

и, как и в операторах READ и WRITE, этот идентификатор не может быть именем объекта, описанного как часть более крупного объекта.

Режим FROM нельзя не указывать для файлов с описателями DIRECT UPDATE и SEQUENTIAL UNBUFFERED UPDATE.

Режим KEY используется в виде

KEY(<выражение>)

и позволяет значение выражения, преобразованное к виду строки знаков, использовать как ключ отыскиваемой записи. Применяется этот режим только для файлов с описателем DIRECT.

Оператор LOCATE применяется при использовании базированной памяти и здесь не рассматривается.

При использовании операторов ввода — вывода в режиме передачи записями необходимо производить явное открытие и закрытие файлов. Для этой цели в языке ПЛ/1 имеются операторы OPEN и CLOSE.

При открытии файла на дисках или на магнитной ленте создаются или проверяются метки и устанавливается связь между программой и файлом. При закрытии файла эта связь прерывается.

Вид оператора OPEN

OPEN <набор режимов открытия>

причем набор состоит из списков режимов, относящихся к открытию какого-нибудь одного файла. Один оператор OPEN может быть использован для открытия любого числа файлов.

Список режимов открытия начинается режимом FILE вида

FILE(<имя файла>)

Вслед за ним может быть указан один из режимов INPUT, OUTPUT или PAGESIZE (<выражение>). Выражение в режиме PAGESIZE определяет число строк на странице, и если этот режим отсутствует, то число строк система определяет на основании данных, записанных в области связи СУПЕРВИЗОРА.

Оператор CLOSE имеет вид

CLOSE <набор режимов закрытия>

причем набор состоит не из списков режимов, а из одного режима на **каждый закрываемый файл**:

FILE(<имя файла>)

4.3. Подготовка программ

Процесс составления программ на исходном языке в настоящей главе предполагается известным, и далее излагаются только те сведения и приемы, которые специфичны для ДОС/ЕС и входящих в ее состав трансляторов. В первую очередь это управление содержанием и объемом выдачи на печать диагностических сообщений, приемы отладки программ, в том числе приемы отыскания ошибок по текстам программ и диагностических сообщений.

4.3.1. Управление содержанием и объемом диагностических сообщений. Средством изменения объема диагностических сообщений во время трансляции программы, ее каталогизации, редактирования и использования (по терминологии ДОС/ЕС — выполнения) являются операторы OPTION и UPSI.

Оператор OPTION с операндом LIST обеспечивает печать программы на исходном языке независимо от того, обнаружены ли в ней ошибки. Программа распечатывается по перфокартам, т. е. для ФОРТРАНа — по операторам. Для ПЛ/1 строка может содержать и несколько операторов и, наоборот, такой оператор, как DECLARE, может располагаться в нескольких строках.

Первая позиция перфокарты в программах на ФОРТРАНе отведена для метки, и наличие в ней пробела означает, что оператор не помещен. Для ПЛ/1, напротив, использование первой позиции считается ошибкой, и, обнаружив ее, транслятор эту позицию игнорирует, сигнализируя об этом программисту печатью 14 звездочек.

Реакция трансляторов на обнаруженные ошибки различна. Так, в Базисном ФОРТРАНе, заканчивая строку, в которой обнаружена ошибка, транслятор в следующей строке указывает ее расположение, печатая точно под ошибкой, пропущенным или искаженным символом знак $. В последующих строках транслятор комментирует обнаруженную ошибку. Так как в одной строке может быть обнаружено несколько ошибок, то перед этим комментарием в скобках указывается номер ошибки. В конце программы печатаются итоговые сообщения, где обычно перечисляются ошибки, которые нельзя отнести к какому-нибудь одному оператору. Обычно таким образом перечисляются неопознанные метки.

Транслятор ПЛ/1, обнаруживая ошибку, не прерывает печатания текста программы, а дает все сообщения вслед за ее текстом, указывая номер неверного оператора. Итоговые сообщения даются в конце.

Следует отметить некоторую неконкретность сообщений транслятора ПЛ/1 об ошибках, что затрудняет их использование. Особенно это относится к сообщениям об ошибках в описании файлов. Кроме того, транслятор объявляет ошибочными все операторы, в которых используются неверно описанные имена, и это может повести по неправильному пути неопытного программиста.

Транслятор ПЛ/1 печатает сообщения об ошибках в виде

<код сообщения><номер сообщения><символ> <текст>

Код сообщения состоит из символов, означающих, что сообщение принадлежит транслятору ПЛ/1 (цифра 5 в первой позиции) и что это сообщение об ошибках, связанных с нарушением синтаксических правил языка и обнаруженных на этапе компиляции (буква Е). Номер сообщения — трехзначное число, полностью определяющее характер ошибки. Текст только разъясняет ее программисту. Символ, идущий вслед за номером сообщения, характеризует тяжесть ошибки: I — для погрешностей, которые только доводятся до сведения программиста; W — транслятор предполагает наличие ошибки, но может продолжать трансляцию; Е — ошибка найдена, но транслятор может ее исправить и продолжать трансляцию; S — для серьезных ошибок, которые транслятор сам исправить не может и, продолжая трансляцию, отменяет последующее редактирование; наконец, Т — когда ошибка настолько грубая, что исключает возможность продолжения трансляции.

В сообщениях об ошибках могут встречаться коды с буквой G во второй позиции, что означает нарушение не синтаксиса языка, а ограничений количественного характера, например, большое количество имен, описанных в одном операторе DECLARE.

Окончание трансляции отмечается сообщением

5W01I SUCCESSFUL COMPILATION (успешная компиляция), если транслятор не обнаружил ошибок, и 5W02I COMPILATION IN ERROR (компиляция с ошибками) при наличии ошибок класса W или Е.

Более грубые ошибки вызывают появление сообщений:

5E03I POSSIBLE ERRORS IN SOURSE PROGRAM (компиляция закончена, возможны ошибки в объектном модуле)

5E02I LINK OPTION RESET (компиляция закончена, редактирование отменено)

5E01I JOBSTEP PL/1 TERMINATED, LINK OPTION RESET (компиляция прекращена, редактирование отменено)

Каждое из этих пяти сообщений является последним сообщением транслятора.

Если программист вместо режима LIST использует режим ERRS, то на SYSLST печатаются только сообщения об ошибках, описанные выше.

В режиме LIST транслятор с Базисного ФОРТРАНа печатает также таблицы распределения памяти как при успешной компиляции, так и в тех случаях, когда компиляция производится при наличии ошибок. Таблица распределения памяти при успешной компиляции состоит из пар *символ*, *адрес*, относящихся к простым переменным, за которыми следуют такие же пары для массивов. Далее в таблице помещается перечень вызываемых подпрограмм и таблица пар *метка*, *адрес*. В конце таблицы указываются объем общей области, объем и адрес загрузки программы. При компиляции с ошибками таблица распределения памяти открывается перечнем неопознанных меток, далее следуют пары *символ*, *адрес* для имен, которые удалось разместить в памяти. Кончается таблица сообщением о прекращении трансляции.

Транслятор ПЛ/1 таблицу распределения памяти выдает при работе в режиме SYM, устанавливаемом с помощью оператора OPTION. Эта таблица состоит из двух частей: *таблицы символических имен* и *таблицы смещений*.

Таблица символических имен содержит:

— исходное имя объекта;

— его внутреннее имя, присвоенное транслятором;

— номер блока, где объявлено имя;

— уровень или глубину блока;

— описатель имени ARRAY, STRUCT., ENTRY или BUILTIN (графа может быть пустой);

— номер уровня (только для структур);

— описатели ARITHM., PICTURE, LABEL, POINTER, STRING или FILE (графа также может быть незаполненной);

— описатели ALIGNED или UNALIGNED, BINARY или DECIMAL, CONST. для констант типа «метка» или VARIAB. для переменных типа «метка»;

— описатели BIT или CHARACTER, FIXED или FLOAT;

— описатель разрядности или пробел;

— описатель класса памяти AUTOM. или STATIC, BASED, PARAM. или DEFIN.;

— описатель области действия EXT. или INT.

В таблице символических имен печатаются сообщения об ошибках, которые могут дополнить описанные выше сообщения транслятора и ускорить отыскание ошибки, так как они печатаются в одной строке с ошибочно объявленным именем. Коды и тексты этих сообщений приводятся ниже.

01 CONFLICTING ATTRIBUTES (противоречивые описатели);

02 TOO MANY DECLARATIONS IN ONE STATEMENT (слишком много частных объявлений в одном операторе DECLARE);

03 PRECISION IS MISSING OR WRONG (не указана или неверно дана длина или разрядность данных);

04 BASE VARIABLE IS BASED OR DEFINED (база сама определена как базированная переменная или через описатель DEFINED);

05 BASE OR POINTER INCORRECT (неправильно объявлен идентификатор базы или указатель);

07 MULTI-DECLARED IDENTIFIER (многократно объявленное имя);

08 RETURN-ATTRIBUTES CONFLICT WITH ENTRY-NAME-ATTR. (описатели типа функции противоречат ее описателям в вызываемой процедуре);

09 INVALID STRUCTURE (неправильное построение структуры);

0A ARRAY TOO LONG (слишком длинный массив);

0B STRUCTURE TOO LONG (слишком длинная структура; глубина вложенности структур не должна превышать 8);

0C UNALIGNED STRUCTURE CONTAINS BIT STRING (структура с описателем UNALIGNED содержит битовую строку);

10 NAME EXCEEDS 31 CHARACTERS (длина идентификатора больше 31 символа);
11 EXTERNAL NAME EXCEEDS 8 CHARACTERS IN LENGTH (длина внешнего имени превышает 8 символов).

Таблица смещений в сочетании с таблицей символических имен позволяет установить размещение в памяти всех программных объектов. Графы таблицы смещений содержат:

— внутреннее имя;

— смещение адреса имени относительно начала области статической или динамической памяти блока, в котором оно объявлено;

— принадлежность имени к классу памяти STATIC или AUTOMATIC; в соответствии с этим отыскивается адрес начала области, в которой оно размещено;

— смещение адреса имени относительно начала модуля (дается только для имен с описателем STATIC).

4.3.2. Идентификация ошибок в программе. Сообщения трансляторов, особенно ФОРТРАНа, а также таблицы распределения памяти и символических имен позволяют хотя и не всегда легким, но надежным способом отыскать ошибки нарушения синтаксиса языка и количественных ограничений транслятора.

Основную часть ошибок нарушения синтаксических правил языка составляют обычно ошибки перфорации, происходящие или из-за нечеткого написания оригинала, или из-за неисправностей устройства подготовки перфокарт. Такие ошибки обычно обнаруживаются сравнительно легко. Труднее отыскать ошибку в описаниях таких данных, как, например, структуры в ПЛ/1 или файлы. Здесь можно рекомендовать в период отладки не соединять большое количество описаний вместе, как это допускает, например, оператор DECLARE в ПЛ/1.

Но успешная компиляция это еще не окончание отладки. Вполне возможно, что при составлении программы или еще раньше, при составлении алгоритма обработки данных, не были предусмотрены все их свойства, могущие привести к непредвиденным реакциям программы. Такие просчеты могут быть обнаружены в процессе эксплуатации программы. В системе программирования ДОС/ЕС предусмотрена выдача диагностических сообщений в период работы программы.

Транслятор Базисного ФОРТРАНа выдает в таких случаях сообщения вида

IJTnnnI

в которых символы IJT идентифицируют сообщения периода работы программы, nnn — номер сообщения, позволяющий установить причину ошибки, а символ I означает, что сообщение предназначено для информации программиста. Большая часть этих сообщений вызывается причинами, приводящими к прекращению работы программы.

Сообщения периода работы программы, сформированные транслятором ПЛ/1, вырабатываются управляющей программой, присоединяемой транслятором к программе пользователя. Так как язык ПЛ/1 позволяет программисту самостоятельно решать, поручать ли обработку ошибок системе или самому строить подпрограммы обработки ошибок, то управляющая программа ПЛ/1 реагирует только на «разрешенные» ошибки. Остальные ошибки программа с помощью оператора ON обрабатывает собственными средствами.

«Разрешенные» ошибки приводят к появлению сообщения вида

5L00I <восьмизначное число> <шестизначное число> ERROR

в котором 5 — признак сообщения транслятора ПЛ/1; L — признак сообщения периода работы программы; I означает, что сообщение предназначено для информации программиста, а выдаваемые числа

<восьмизначное число> : : = ccdddddd
<шестизначное число> : : = aaaaaa

где

cc : : = <код ошибки>
aaaaaa : : = <абсолютный адрес команды, породившей ошибку>

Если причиной ошибки является файл, то

dddddd : : = <адрес блока ввода — вывода>

В противном случае в этих позициях печатаются нули.

Причины ошибок, вызывающих появление того или иного кода ошибки «cc», т. е. связь кодов сообщений программы и вызывающих их ошибок для ПЛ/1, приводятся в табл. 4.3.

Таблица 4.3

Вторая цифра кода	Первая цифра кода сообщения								
	0	1	2	3	4	5	6	7	8
	Программные прерывания	Машинные прерывания	Ошибки организации	Ошибки при обращении к математическим функциям		Ошибки при обращении к арифметическим функциям	Ошибки при вводе — выводе		
				обычной длины	двойной длины				
0	—	—	—	Аргумент функции SQRT (X)<0	Аргумент функции SQRT (X)<0	Аргумент Y функции MOD (X, Y) равен нулю	—	Ошибка при размещении по ключу	Запись в файле по ключу не найдена
1	Переполнение (плавающая точка)	Обычная операция	Мало места для DSA	Абсолютное значение аргумента синуса и косинуса больше 2^{18}	Абсолютное значение аргумента синуса и косинуса $>2^{50}$	То же	Несовместимы списки данных и форматов	Переполнена область индекса цилиндра	Переполнена область переполнения
2	Исчезновение мантиссы	Привилегированная операция	Неверна метка в GOTO	Абсолютное значение аргумента тангенса больше 2^{18}	Абсолютное значение аргумента тангенса $> 2^{50}$	Для функции MOD (X, Y) или $\mid X/Y \mid <$ $<5.4.20^{78}$ или же $X/Y >$ $>7.2.10^{75}$	Выход за пределы строки в режиме STRING	Переполнена область главного индекса	Переполнена основная область данных
3	Деление на нуль	Команда „выполнить"	Неверный вызов главной процедуры	Значение функции TAN (X) → ∞	Значение функции TAN (X) →∞		Форматы печати заданы для файла без описателя PRINT	—	Запись с таким ключом уже имеется в файле
4	Переполнение (фиксированная точка)	Защита памяти	—	Оба аргумента функции TAN (X, Y) равны нулю	Оба аргумента функции TAN (X, Y) равны нулю	—	Неправильное GET или PUT	—	Ошибка в ключе

5	Потеря старших разрядов (фиксированная точка)	Адресация	—	Аргумент гиперболического синуса или косинуса превышает 174.6	Аргумент гиперболического синуса или косинуса превышает 174.6	—	Неправильно задан оператор для файла CONSECUTIVE BUFFERED	—	—
6	Недопустимое преобразование строки знаков	Спецификация	—	Аргумент функции e^x превышает 174.6	Аргумент функции e^x превышает 174.6	—	То же для CONSECUTIVE UNBUFFERED	—	—
7	—	Данные	—	Аргумент функции ATANH (X) больше 1	Аргумент функции ATANH(X) > 1	—	То же для REGIONAL	—	—
8	—	—	—	Аргумент логарифмической функции $\leqslant 0$	Аргумент логарифмической функции $\leqslant 0$	—	—	—	—
9	Причина не установлена	—	—	В выражении x^y $x = 0$, $y \leqslant 0$	В выражении x^y $x = 0$, $y \leqslant 0$	—	Режимы PAGE или SIZE для файла без описателя PRINT	—	—
A	Достижение конца файла	—	—	В выражении x^y оба аргумента нули	В выражении x^y оба аргумента нули	—	Неправильно задан оператор для файла INDEXED DIRECT	—	—
C	Ошибки передачи файла	—	—	—	—	—	То же, для INDEXED SEQUENTIAL	—	—
I	Неверный ключ	—	—	—	—	—	—	—	—
E	Неверная длина записи	Значность	—	—	—	—	—	—	—

Примечание. Прочерком отмечены коды, не используемые транслятором ПЛ/1 ДОС/ЕС.

Так как использование абсолютного адреса команды, послужившей причиной ошибки, затруднительно, программист пользуется средством перевода этого адреса в номер оператора, преобразованного транслятором в эту команду. Этим средством является оператор UPSI с операндом 01

// UPSI 01

Поместив его перед оператором

// EXEC <имя транслятора>

программист получит уже сообщения вида
5L00I ccdddddd aaaaaa ERROR STATEMENT

NUMBER nnnn

и сразу будет знать место ошибочного оператора в программе, написанной на исходном языке.

4.3.3. Построение оверлейных программ. Из языков высокого уровня ДОС/ЕС многофазные программы позволяет строить только ПЛ/1. РЕДАКТОР лишь настраивает фазы на заданные программистом (прямо или косвенно) адреса памяти. Управление же процессом смены фаз должно осуществляться самой программой. В ПЛ/1 для этого имеется модификация оператора вызова процедур CALL OVERLAY.

При необходимости вызвать процедуру, не находящуюся в оперативной памяти, программист использует два оператора вызова

CALL OVERLAY('<имя фазы>');

CALL<имя процедуры> (список фактических параметров)

Эти операторы размещаются в процедуре, которую можно назвать управляющей и которая должна находиться в корневой фазе программы (см. гл. 6). Вызов фаз ничем не ограничен, и он может зависеть от характера получаемых в процессе работы программы промежуточных результатов.

5. Библиотекарь

Библиотечное хозяйство является неотъемлемой частью ДОС/ЕС, так как большинство ее компонент хранится в *библиотеке абсолютных модулей*. Некоторые

программы ДОС могут храниться и в других библиотеках. Три уже известные нам библиотеки: CL, RL и SL вместе взятые составляют так называемый *резидентный файл*, устанавливаемый при работе машины на устройстве SYSRES. Эти библиотеки называются *системными* в отличие от *личных*, которые программист может создавать на других дисковых устройствах.

В резидентном файле библиотека CL всегда присутствует, а двух других может и не быть.

5.1. Структура библиотек и резидентного файла ДОС/ЕС

Каждой из библиотек в резидентном файле выделяется целое число цилиндров дискового пакета, а оглавлениям библиотек — целое число дорожек на первом или первых из выделенных для библиотеки цилиндрах. Выделяемые для библиотек цилиндры и дорожки для их оглавлений должны выделяться подряд (оглавления начинаются с нулевой дорожки первого выделенного библиотеке цилиндра).

Область библиотеки абсолютных модулей должна начинаться с нулевого цилиндра пакета. Нулевой цилиндр выделяется под системные материалы: выборки из оглавления CL для наиболее часто употребляемых программ, в том числе транзитов СУПЕРВИЗОРА, ЛСУВВ и т. п.

Непосредственно к последней из библиотек примыкает *цилиндр меток* — область, в которой хранятся данные об используемых программистом файлах. Остаток дискового пакета за цилиндром меток программист может использовать по своему усмотрению.

Системные материалы	Цилиндр 0
Оглавление CL	⩾1 полных дорожек цилиндра 0
CL	⩾1 полных цилиндров
[Оглавление RL]	⩾1 полных дорожек
[RL]	⩾1 полных цилиндров
[Оглавление SL]	⩾1 полных дорожек
[SL]	⩾1 полных цилиндров
Цилиндр меток	1 полный цилиндр
[Область пользователя]	Если остается место
Оглавление тома	Последний цилиндр

Рис. 5.1. Структура резидентного тома.

На рис. 5.1 показан состав резидентного файла системы и резидентного тома в целом.

Библиотека абсолютных модулей (CL) состоит из абсолютных модулей или *фаз*. Фаза — это программа или ее часть, полностью подготовленная к выполнению.

Если программа состоит из нескольких фаз, то желательно присваивать им имена, у которых совпадали бы первые четыре символа. Это не является обязательным, но ускоряет подготовку задачи к решению.

В состав CL входит *оглавление*, в котором для каждой фазы приводятся:

— имя (1—8 символов);

— адрес фазы на диске: т. е. номера цилиндра, дорожки, номер первой записи на дорожке, отведенной этой фазе;

— количество записей, из которых состоит текст фазы (все записи в CL имеют фиксированную длину 1728 байтов);

— количество байтов фазы в последней записи, которая может быть неполной;

— адрес в оперативной памяти, начиная с которого должна быть загружена фаза;

— адрес точки входа в фазу, т. е. первой ее команды (началом фазы может быть не программный, а какой-либо табличный материал).

Размещение фаз в CL в отличие от размещения материалов в других библиотеках ДОС/ЕС выполняется не БИБЛИОТЕКАРЕМ, а РЕДАКТОРОМ. Все остальные библиотечные функции как в CL, так и в других библиотеках выполняет БИБЛИОТЕКАРЬ. Размещаются фазы в CL в порядке их засылки и для других функций отыскиваются по имени. Поэтому имя каждой фазы должно быть уникальным. Начало оглавления, включающее сведения о транзитах СУПЕРВИЗОРА, называется *транзитным оглавлением*.

Библиотека объектных модулей (RL) состоит из программ, предназначенных для обработки РЕДАКТОРОМ. Эти программы переведены с языков программирования в коды машины, но адреса команд, данных и связей с другими такими же программами остались условными. Поэтому наряду с основным содержанием в таких программах (объектных модулях) имеется информация, позволяющая РЕДАКТОРУ преобразовывать их в абсолютные модули.

Каждый объектный модуль является результатом трансляции какого-либо исходного модуля, написанного

на одном из языков программирования ДОС/ЕС: Базисном ФОРТРАНе, ФОРТРАНе-IV, ПЛ/1, РПГ или языке АССЕМБЛЕРА. Достигается это с помощью оператора УЗ

// EXEC <имя транслятора>

где

<имя транслятора> : : = FORTRAN|FFORTRAN|PL/I|RPG|ASSEMBLY|FCOBOL...

Оглавление библиотеки RL для каждого объектного модуля содержит запись со следующей информацией:

— имя (1—8 символов);

— адрес (номера цилиндра, дорожки, первой записи);

— количество записей (так как записи в RL имеют переменную длину (200—300 байтов), то для каждой из них известно число содержащихся в ней байтов и необходимость указания числа байтов в последней записи отпадает);

— версия и модификация модуля, позволяющие учитывать количество вносимых в модуль изменений.

Модули в RL могут быть и отдельно протранслированными частями одной и той же программы; в этом случае для удобства оперирования с ними их имена имеют общие первые три символа. Но имя каждого модуля в RL, как и в CL, должно быть уникальным.

Модуль в RL является наименьшей единицей, которой оперирует БИБЛИОТЕКАРЬ. Но ниже, на этапе редактирования, мы будем рассматривать и части модуля — *программные секции.*

Библиотека исходных модулей (SL) состоит из программ, написанных программистами на языках программирования ДОС/ЕС. Единицей хранения в SL является *исходный модуль* или *книга*. Книга, вообще говоря, не обязательно должна быть законченной программой. Например, в трансляцию можно пустить текст, составленный из нескольких книг. Можно в SL хранить также произвольную текстовую информацию. Книги в SL могут объединяться в *подбиблиотеки*, причем полное имя книги имеет вид

<символ>.<имя книги>

где символ идентифицирует подбиблиотеку.

В оглавлении библиотеки исходных модулей для каждой содержащейся в ней книги имеется запись следующего состава:

— полное имя книги;

— ее адрес (номера цилиндра, дорожки, первой записи);

— количество записей, составляющих книгу;

— версия и модификация книги.

Книги в SL и записи в оглавлении этой библиотеки идут в порядке их поступления и никакого упорядочения по подбиблиотекам в них нет. Все операции над книгами, подбиблиотеками и библиотекой в целом выполняет БИБЛИОТЕКАРЬ. Одна из функций БИБЛИОТЕКАРЯ — *обновление* позволяет оперировать и с отдельными частями книг — строками (для ФОРТРАНа и языка АССЕМБЛЕРА понятие строки и оператора совпадают).

Личные библиотеки устроены таким же образом, как и системные, но находятся на другом или на других дисковых пакетах и каждая представляет собой отдельный файл. Имеется возможность создания в системе любого количества личных библиотек.

5.2. Состав БИБЛИОТЕКАРЯ и его функции

Основные функции БИБЛИОТЕКАРЯ показаны на рис. 5.2. Выполняются они программами, хранящимися в CL и имеющими имена:

MAINT — для всех функций корректировки;
CSERV — для всех сервисных функций в отношении CL;
RSERV — то же в отношении RL;
SSERV — то же в отношении SL;
JSERV — для пересылки информации из SL на трансляцию;
DSERV — для печати оглавлений библиотек;
CORGZ — для создания и копирования библиотек и резидентного файла в целом.

Выбор функции БИБЛИОТЕКАРЯ, а иногда и библиотеки, к которой относится функция, достигается использованием *управляющих операторов* БИБЛИОТЕКАРЯ, которые вводятся с SYSIPT. Именно поэтому даже при отсутствии других вводимых данных вслед за опе-

раторами БИБЛИОТЕКАРЯ во входном потоке SYSIPT должен следовать оператор УПРАВЛЕНИЯ ЗАДАНИЯМИ:

/*.

означающий конец вводного файла.

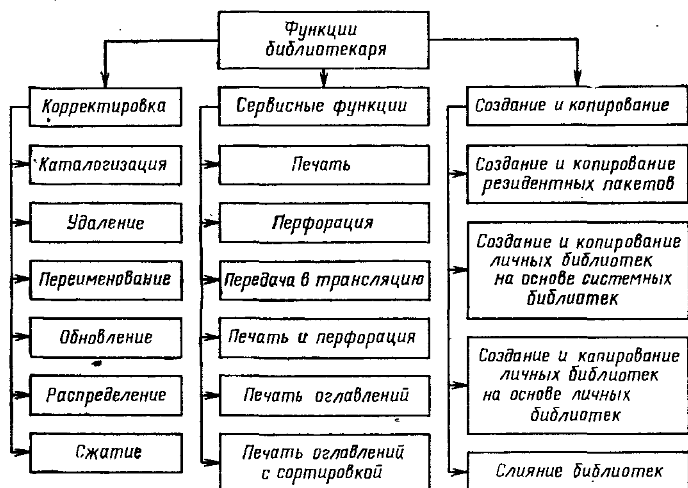


Рис. 5.2. Функции БИБЛИОТЕКАРЯ ДОС/ЕС.

5.3. Формирование задания для БИБЛИОТЕКАРЯ

Обращение к БИБЛИОТЕКАРЮ может быть как первым, так и одним из последующих шагов задания. В первом случае набор операторов УЗ начинается оператором

// JOB <имя задания>

БИБЛИОТЕКАРЬ пользуется рядом системных ЛУВВ и устройств ввода — вывода программиста. Если среди них (кроме SYSRES) применяются диски, то ДОС следует сообщать информацию о метках используемых файлов, для чего необходимы операторы УЗ

// DLBL <операнды>
// EXTENT <операнды>,

в которых среди прочих данных надо указывать имена системных файлов БИБЛИОТЕКАРЯ (см. табл. 5.1).

Таблица 5.1

Имя ЛУВВ	Имя файла	Имя ЛУВВ	Имя файла
SYSCLB	IJSYSCL	SYSLNK	IJSYSLN
SYSRLB	IJSYSRL	SYS000	IJSJSPS
SYSSLB	IJSYSSL	SYS001	IJSYSPR
SYSIPT	IJSYSIN	SYS002	JSYSRS
SYSRDR		SYS003	IJSYSRC
SYSLST	IJSYSLS	SYS009	IJSYS09
SYSPCH	IJSYSPH		

Необходимые назначения УВВ делаются, как обычно, с помощью оператора УЗ

// ASSGN <операнды>

Далее идет оператор EXEC с именем необходимой программы БИБЛИОТЕКАРЯ

// EXEC <имя>

являющейся концом шага задания.

Используемая им информация берется уже с SYSIPT и состоит из:

— одного или нескольких управляющих операторов БИБЛИОТЕКАРЯ;

— входных данных к некоторым из них;

— оператора УЗ /*, указывающего конец файла на SYSIPT.

Если шаг задания является последним, то на SYSRDR помещают оператор

/&

Формат управляющих операторов БИБЛИОТЕКАРЯ подобен формату АССЕМБЛЕРА:

{<пробел>}...<код оператора>{<пробел>}...
...<операнды>

В поле операндов не должно быть пробелов, и они не должны выходить за пределы 71-й позиции перфокарты.

5.4. Корректировка

Возможности БИБЛИОТЕКАРЯ в части корректировки библиотек показаны в табл. 5.2. Способы каталогизации в CL, выполняемой не БИБЛИОТЕКАРЕМ, а РЕДАКТОРОМ, излагаются в гл. 6.

Все функции корректировки выполняются с помощью программы MAINT и детализируются указанными в табл. 5.2 операторами. За одно выполнение программы MAINT могут быть обслужены одна, две или три разные библиотеки и для каждой из них могут быть выполнены любые перечисленные в табл. 5.2 функции над произвольным числом элементов библиотек:

Таблица 5.2

Функции корректировки	Формат управляющих операторов БИБЛИОТЕКАРЯ для		
	CL	RL	SL
Каталогизация	Нет	CATALR <модуль> [,V.M] [,SYSLNK]	CATALS <имя> [,V.M[,C]]
Удаление	DELETC <список>	DELETR <список>	DELETS <список>
Переименование	RENAMC <список>	RENAMR <список>	RENAMS <список>
Обновление	Нет	Нет	UPDATE <имя> [,<имя>] [,V.M] [,nn] <операторы>
Сжатие	CONDS <список>		
Установление параметра автоматического сжатия	CONDL <список>		
Распределение	ALLOC <список>		

5.4.1. Каталогизация в RL и SL. Для каталогизации модуля в RL требуется оператор БИБЛИОТЕКАРЯ

CATALR <имя> [,V.M] [,SYSLNK]

где

<имя> — имя каталогизируемого модуля, состоящее из 1—8 буквенно-цифровых символов, первый из которых не должен быть звездочкой (*)

V.M указывает *версию* и *модификацию* модуля, причем $0 \leqslant V \leqslant 127$, $0 \leqslant M \leqslant 255$;

SYSLNK означает, что модуль с таким именем надо искать на SYSLNK, а если этого операнда нет, то модуль находится на SYSIPT. Любому модулю на SYSLNK предшествует оператор вида

CATALR <имя>

(операнд не применяется для модулей, полученных трансляцией с ПЛ/1 и РПГ).

Пример 23. Каталогизировать модуль ABC235, находящийся на SYSIPT. В качестве устройств SYSRDR и SYSIPT назначается одно и то же устройство.

Решение.

```
// JOB ПРИМ23
// EXEC MAINT
   CATALR ABC235
   <карты модуля>
/*
/&
```

Модулю присваиваются нулевые значения версии и модификации.

Пример 24. Исходный модуль на ФОРТРАНе превратить в объектный и каталогизировать под именем XYZA4.

Решение.

```
// JOB ПРИМ24
// OPTION LINK
* ОБЕСПЕЧИВАЕТ ПОМЕЩЕНИЕ РЕЗУЛЬТАТА ТРАНСЛЯЦИИ
* НА SYSLNK ЧТОБЫ ПЕРЕД МОДУЛЕМ ОБЕСПЕЧИТЬ ЗА-
* ПИСЬ НА SYSLNK ОПЕРАТОРА CATALR, ПИШЕМ ОПЕРА-
* ТОР РЕДАКТОРА:
   INCLUDE
   CATALR XYZA4
/*
* INCLUDE НА SYSRDR, А CATALR И КОНЕЦ ФАЙЛА НА
* SYSIPT.
// EXEC FORTRAN
   <карты исходного модуля>
/*
// EXEC MAINT
   CATALR XYZA4,0.0,SYSLNK
/*
/&
```

При нежелании пользоваться оператором РЕДАКТОРА INCLUDE, а также при необходимости получить *смешанный* модуль (см. ниже гл. 6) и каталогизировать его поступают следующим образом.

Пример 25. Набор управляющих перфокарт, реализующих это задание, будет иметь вид

```
// JOB ПРИМ25
// OPTION DECK
// EXEC PL/I
      <исходный модуль на п/к>
/*
// PAUSE ВЫНУТЬ П/К ИЗ SYSPCH,
* ДОБАВИТЬ ПРИЛОЖЕННЫЕ ПЕРЕД
* ВЫДАННЫМИ И ПОЛОЖИТЬ В SYSIPT
// EXEC MAINT
      CATALR AXBYCZ,0.1
      <модуль, который оператор должен успеть переложить из
      SYSPHC в SYSIPT>
/*
/&
```

Если никаких карт добавлять не нужно, то все эти операции можно сделать путем переназначений.

Поместить перед текстом модуля управляющий оператор БИБЛИОТЕКАРЯ

CATALR <имя модуля>

можно с помощью оператора УПРАВЛЕНИЯ ЗАДАНИЯМИ

// UPSI nnnnnnnn

Этот оператор помещает в 23-й байт области связи двоичное число, являющееся его операндом. Трансляторы ДОС/ЕС реагируют на нулевой (старший) бит 23-го байта и помещают перед формируемым модулем оператор БИБЛИОТЕКАРЯ, причем в качестве имени модуля выбирается:

— транслятором ПЛ/1 метка основного входа в программу или подпрограмму;

— транслятором ФОРТРАНа для процедуры ее имя, а для основной программы слово FORTMAIN. Поэтому в последнем случае в целях обеспечения уникальности имени модуля непосредственно после каталогизации следует его переименовать.

Ниже приводится пример подобной каталогизации.

Пример 26. Каталогизировать объектный модуль после трансляции с использованием магнитной ленты.

Решение.

```
// JOB ПРИМ26
// OPTION DECK
```

```
// ASSGN SYSPCH,X'240'
* 240—АДРЕС НМЛ,ВЫБРАННОГО ДЛЯ SYSPCH.
// MTC REW,SYSPCH
* ОПЕРАТОР MTC С ОПЕРАНДОМ REW ОБЕСПЕЧИВАЕТ ПЕ-
* РЕМОТКУ ЛЕНТЫ НА НАЧАЛО.
// UPSI 1
* В ОПЕРАТОРЕ UPSI НУЛИ ПОСЛЕ ПОСЛЕДНЕЙ ЕДИНИ-
* ЦЫ МОЖНО НЕ ПИСАТЬ.
// EXEC FORTRAN
   <перфокарты с исходным модулем>
/*
// MTC WTM,SYSPCH
* ОПЕРАНД WTM ОПЕРАТОРА MTC ОБЕСПЕЧИВАЕТ ЗАПИСЬ
* ВТОРОГО МАРКЕРА ПОСЛЕ ПОСЛЕДНЕГО ФАЙЛА.
// MTC PEW,SYSPCH
// ASSGN SYSPCH,UA
* SYSPCH СТАНОВИТСЯ НЕДОСТУПНЫМ.
// ASGN SYSIPT,X'240'
// EXEC MAINT
* CATALR МОЖНО НЕ ПИСАТЬ, ТАК КАК ЭТОТ ОПЕРАТОР
* ВПИСАН ТРАНСЛЯТОРОМ ПЕРЕД МОДУЛЕМ.
/&
```

Каталогизация в библиотеку исходных модулей позволяет добавлять в нее исходные модули — книги, содержащие:

— набор операторов на любом из языков ДОС/ЕС, не обязательно являющийся законченным блоком программы;

— макроопределения языка АССЕМБЛЕРА;

— набор управляющих операторов ДОС/ЕС, в том числе /* и /&;

— произвольные тексты, поступающие в виде файлов на перфокартах.

Источник информации в первых двух случаях — SYSIPT или SYS009, а в остальных — только SYS009, так как иначе операторы /* и /& могут быть восприняты не как составные части вводимого файла, а как операторы УЗ.

Перед каталогизацией книга оформляется в виде

```
BKEND <операнды>
<текст книги>
BKEND
```

У заглавного BKEND

```
<операнды> : : = [<имя книги>] [,SEQNCE]
[,<счетчик>] [,CMPRSD]
```

Все эти операнды не обязательны.

Способ описания книги был изложен выше. Остальные операнды означают следующее:

— наличие операнда SEQNCE обеспечивает контроль возрастания содержимого колонок 77—80 колоды, применяемого обычно для нумерации перфокарт;

— операнд <счетчик> задает число карт в книге, включая карты BKEND, и препятствует ее каталогизации при обнаружении расхождений;

— операнд CMPRSD указывает, что книга задана в сжатом формате.

В конечном операторе BKEND текст в поле операндов не используется.

Если каталогизируемая книга является макроопределением АССЕМБЛЕРА, то она оформляется в виде

```
MACRO
<текст книги>
MEND
```

Формат оператора каталогизации в SL следующий:

— если книга находится на SYSIPT

```
CATALS <имя книги>[,V.M[,C]]
```

— если книга находится на SYS009

```
CATALS1 <имя книги>[,V.M[,C]]
```

Здесь новым является только появление операнда C, который обеспечивает при модификации книги автоматическое увеличение кода V.M, но при этом от программиста требуется знание текущего его значения. При отсутствии операнда C контроль и увеличение кода V.M не производятся.

Пример 27. Каталогизировать книгу LMN001, находящуюся на SYSIPT, в подбиблиотеку ПЛ/1 (символ P). В книге 250 ненумерованных перфокарт, формат несжатый. Значения V=0, M=63 надо модифицировать. В качестве устройств SYSRDR и SYSIPT стандартно назначено устройство ввода с перфокарт X'00C'.

Решение.

```
// JOB ПРИМ27
// EXEC MAINT
    CATALS P.LMN001,0.63,C
    BKEND P.LMN001,250
    <перфокарты исходного модуля>
    BKEND
/*
/&
```

Операторы BKEND затрудняют использование библиотеки SL для хранения программ перед их трансляцией. Непосредственная выдача исходного модуля на перфокарты с удалением из него вручную карт BKEND задерживает процесс обработки. Пересылка же модуля на диски или ленту с помощью программы перфорации с назначением в качестве SYSPCH дисковых или ленточных устройств также не решает вопроса полностью, так как в этом случае перед каждой записью программа пересылки формирует дополнительный байт — признак выбора кармана перфоратора при перезаписи модуля с устройства, игравшего роль SYSPCH, на реальный перфоратор.

Для преодоления этого затруднения в состав программ БИБЛИОТЕКАРЯ введена программа JSERV, позволяющая производить фиктивное перфорирование без формирования перед записями этого байта.

Относительно каталогизации модулей в личные библиотеки RL и SL следует заметить, что при наличии назначения устройств для личной библиотеки каталогизация производится в личную библиотеку, а при отсутствии назначения — в системную.

Пример 28. Исходный модуль PQR12 на языке ФОРТРАН (подбиблиотека F) каталогизировать в личную библиотеку SL, находящуюся на дисковом пакете № 123456, цилиндр 35. Пакет находится на устройстве X'107'. Номера версии и модификации и число перфокарт в книге игнорировать.

Решение.

```
// JOB ПРИМ28
// DLBL IJSYSSL
* ИМЯ ФАЙЛА ВЗЯТО ИЗ ТАБЛ. 5 1
// EXTENT SYSSLB,123456,1,0,350,550
// ASSGN SYSSLB, X'107'
// EXEC MAINT
   CATALS F.PQR12
   BKEND F.PQR12
   <книга>
   BKEND
/*
/&
```

5.4.2. Удаление. *Удаление* является операцией, которую БИБЛИОТЕКАРЬ может выполнить по отношению к содержимому любой из системных или личных библиотек ДОС/ЕС. Возможные операнды и объекты удаления из той или иной библиотеки приведены в табл. 5.3.

Таблица 5.3

Наименование библиотеки	Объекты удаления	Формат	
		оператора	операнда
CL	Фаза Программа	DELETC	<имя фазы> <4 символа>. ALL
RL	Модуль Программа Вся библиотека	DELETR	<имя модуля> <3 символа>. ALL ALL
SL	Книга Подбиблиотека Вся библиотека	DELETS	<имя книги> <символ подбиблиотеки>. ALL ALL

Процедура удаления не производит физического удаления объекта из библиотеки. Его данные удаляются только из оглавления библиотек, делая его недоступным для использования, место же в библиотеке он продолжает занимать.

В табл. 5.3. приведен вид одного операнда управляющего оператора удаления. В общем случае допускается написание

<оператор> <список>

где согласно табл. 5.3.

<оператор> : : =DELETC|DELETR|DELETS
<список> : : =<операнд>|<операнд>,<список>

Такое определение не ограничивает длины списка и поэтому нуждается в уточнении: список должен укладываться в 71 позицию перфокарты.

За одно выполнение программы MAINT может быть обработано любое число одинаковых или разных операторов типа DELET, а также других операторов БИБЛИОТЕКАРЯ. Но одновременно программа MAINT может оперировать только с тремя библиотеками, и при необходимости работать с системными и личными библиотеками одного типа требуется делать переназначения УВВ. Ниже приведено несколько примеров удаления модулей, программ и библиотек.

Пример 29. Удалить модуль A127B из системной библиотеки RL и программу XYZ в составе пяти модулей XYZ1, XYZ2,..., XYZ5 из личной библиотеки RL, находящейся на пакете № 231576, установленном на устройстве X'108' (цилиндры 10—14).

Решение.

```
// JOB ПРИМ29
// EXEC MAINT
   DELETR A127B
/*
// DLBL IJSYSRL
* ИМЯ ФАЙЛА ОПЯТЬ ВЗЯТО ИЗ ТАБЛ.5.1.
// EXTENT SYSRLB,231576,1,0,100,50
// ASSGN SYSRLB,X'108'
// EXEC MAINT
   DELETR XYZ.ALL
/*
/&
```

Пример 30. В целях расширения библиотеки CL удалить полностью библиотеки RL и SL и транслятор с Базисного ФОРТРАНа в CL.

Решение.

```
// JOB ПРИМ30
// ASSGN SYSRLB,UA
// ASSGN SYSSLB,UA
* ЗАЩИЩАЕМ ОТ УДАЛЕНИЯ ЛИЧНЫЕ БИБЛИОТЕКИ
// EXEC MAINT
  DELETC FORT.ALL
  DELETR ALL
  DELETS ALL
/*
/&
```

Пример 31. Каталогизировать с перфокарт в RL модуль S12T34 и удалить из SL подбиблиотеку D.

Решение.

```
// JOB ПРИМ31
// EXEC MAINT
   CATALR S12T34
   <карты модуля>
  DELETS D.ALL
/*
/&
```

5.4.3. Переименование. *Переименование* — это замена старого имени объекта новым. После переименования система перестает распознавать старое имя. Но если имя, которое должно стать новым, уже есть в библиотеке, то переименование не выполняется, а программисту выдается сообщение о наличии объекта с этим именем. Переименование чаще всего приходится применять при

каталогизации модулей, получаемых трансляцией с ФОРТРАНа, так как этот транслятор основной программе всегда присваивает имя FORTMAIN.

В отличие от удаления переименование может касаться только элементарных объектов — модулей, что и показано в табл. 5.4.

Таблица 5.4

Наименование библиотеки	Объем переименования	Формат	
		оператора	операндов
CL	Фаза	RENAMC	<старое имя фазы>, <новое имя фазы>
RL	Объектный модуль	RENAMR	<старое имя модуля>, <новое имя модуля>
SL	Книга	RENAMS	<старое имя книги>, <новое имя книги>

В общем случае оператор переименования имеет вид

<оператор> <список>

где согласно табл. 5.4

<оператор> : : =RENAMC|RENAMR|RENAMS

а

<список> : : =<операнд>|<операнд>, <список>

причем

<операнд> : : =<старое имя>,<новое имя>

что и расшифровано для каждой библиотеки в табл. 5.4.

Как и для случая удаления, список должен иметь длину, обеспечивающую размещение всего текста оператора в пределах 71-й колонки перфокарты.

Для библиотеки SL при переименовании

<старое имя> : : =<символ подбиблиотеки>.<собственно имя книги>

<новое имя> : : =<символ подбиблиотеки>.<собственно имя книги>|<собственно имя книги>

Символ подбиблиотеки изменять не разрешается, и в новом имени его можно не указывать.

Остальные замечания относительно использования операторов переименования полностью совпадают с таковыми для случая удаления.

Пример 32. Переименовать фазу BLAC3 в IJHKL, модули ABC45 и ABC46 в личной библиотеке RL и XYZ5 и XYZ6, модуль PQ23A в системной библиотеке RL в AB23P и книгу F.1235 в личной библиотеке SL в F.A1235.

Личная библиотека RL находится на НМД X'106', на дисковом пакете № 987654, цилиндры 50—69 включительно, а личная библиотека SL там же, на цилиндрах 70—99.

Решение.

```
// JOB ПРИМ32
// DLBL IJSYSRL
// EXTENT SYSRLB, 987654,1,0,500,200
// DLBL IJSYSSL,
// EXTENT SYSSLB,987654,1,0,700,300
// ASSGN SYSRLB,X'106'
// ASSGN SYSSLB,X'106'
// EXEC MAINT
   RENAMC BLAC3,IJHKL
   RENAMR ABC45,XYZ5,ABC46,XYZ6
   RENAMS F.1235, F.A1235
/*
// ASSGN SYSRLB,UA
// EXEC MAINT
   RENAMR PQ23A,AB23P
/*
/&
```

Пример 33. Транслировать два исходных модуля с языка ФОРТРАН и присвоить им в RL имена ABC1 и ABC2, имея в виду использовать их как части одной программы. Использовать НМЛ с адресом X'20C'

Решение.

```
// JOB ПРИМ33
// OPTION DECK
// MTC REW,X'20C'
* МАГНИТНАЯ ЛЕНТА ПРИВЕДЕНА В ИСХОДНОЕ ПОЛОЖЕ-
* НИЕ
// ASSGN SYSPCH,X'20C'
// UPSI 1
* ТЕПЕРЬ ТРАНСЛЯТОР ПЕРЕД ОБЪЕКТНЫМ МОДУЛЕМ
* ВСТАВИТ ОПЕРАТОР CATALR FORTMAIN
// EXEC FORTRAN
   <карты исходного модуля 1>
/*
// ASSGN SYSPCH,UA
// ASSGN SYSIPT,X'20C'
// MTC REW,SYSIPT
// EXEC MAINT
// RESET SYSIPT
// ASSGN SYSPCH,X'20C'
// MTC REW,SYSPCH
```

```
// EXEC MAINT
   RENAMR FORTMAIN,ABC1
/*
// EXEC FORTRAN
   <карты исходного модуля 2>
/*
// ASSGN SYSPCH,UA
// ASSGN SYSIPT,X'20C'
// MTC REW,SYSIPT
// EXEC MAINT
// RESET SYSIPT
// EXEC MAINT
   RENAMR FORTMAIN,ABC2
/*
/&
```

5.4.4. Обновление. Операция *обновления*, применяемая только в SL, позволяет:

— заменять, добавлять и удалять строки в книге;

— изменять в книге нумерацию перфокарт, занимающую колонки 73—76 карты;

— перенумеровывать строки внутри книги;

— изменять номера версии и модификации книги;

— присваивать книге параметр C, что при дальнейших обновлениях позволяет автоматически изменять значения V и M;

— удалять параметр C, исключая тем самым процесс обновления значений версии и модификации этой книги;

— копировать книгу.

Операция обновления очень полезна на этапе отладки программы. Для выполнения этой операции после

//EXEC MAINT

непосредственно или после других операторов БИБЛИОТЕКАРЯ ставится оператор

UPDATE<изменение>[,<дополнительные параметры>]

обязательно с пробелом в первой позиции.

Операнды оператора UPDATE определяются ниже

<изменение> : : =<результат изменения>[,<объект изменения>]

<результат изменения> : : =<символ>.<имя изменяемой книги>

<символ> : : =<символ подбиблиотеки>

Наименование книги при обновлении сохраняется:

<объект изменения> : :=<снмовол>.<имя, под которым может быть сохранена старая редакция книги>

Имя книги в старой редакции должно отличаться от исходного. Этот операнд указывается на случай, если при модификации есть опасность уничтожения верного текста. Символ подбиблиотеки при таком переименовании объекта изменения сохраняется.

Отсутствие среди операндов оператора UPDATE объекта изменения приводит к удалению его метки из оглавления библиотеки, хотя физически он до выполнения операции сжатия в ней остается:

<дополнительные параметры> : : =<параметр версии>[,<параметр нумерации>]

<параметр версии> : : =V·M|<пусто>

где V — номер версии, $0\leqslant V\leqslant 127$; M — номер модификации, $0\leqslant M\leqslant 255$.

При очередном обновлении коды V·M рассматриваются как единое 15-разрядное двоичное число, которое после обновления увеличивается на единицу, если каталогизация книги производилась с параметром С (см. п. 5.4.1). Значения кодов V·M в операторе UPDATE должны совпадать с их текущими значениями для обновляемой книги, в противном случае обновление не производится:

<параметр нумерации> : : =<цифра≠0>|10|NO|<пусто>.

Обозначается этот параметр через пп. Перенумерация при обновлении, в котором указано значение пп, идет от 0000 с шагом пп (например, при пп=3 имеем 0000, 0003, 0006, 0009 и т. д.). При пп==<пусто> подразумевается пп=1. При пп=NO сохраняется старая нумерация.

Существо вносимых в книгу изменений должно быть описано в операторах конкретизации, помещаемых непосредственно за оператором UPDATE. Каждый из этих операторов размещается на отдельной перфокарте, начиная с первой ее позиции, в которой ставится закрывающая скобка. Во второй позиции оставляется пробел, а далее помещается код оператора и дальнейший текст,

Если вслед за оператором конкретизации располагается вносимая в модуль информация, то окончание ее БИБЛИОТЕКАРЬ обнаруживает по появлению в первой позиции признака начала нового оператора конкретизации — закрывающей скобки.

Перенумерацию строк книги можно выполнить и без внесения в нее изменений. Для этого достаточно записать

UPDATE<символ подбиблиотеки>.<имя книги>,

V.M, nn

задав необходимое nn и не помещая вслед за оператором UPDATE никаких операторов конкретизации, кроме оператора END.

Ниже описаны применяемые БИБЛИОТЕКАРЕМ операторы конкретизации:

<оператор конкретизации> : : = <оператор добавления> | <оператор удаления> | <оператор замены> | <оператор перенумерации> | <оператор конца>

Формат *оператора добавления*

)ADD N

где) — закрывающая круглая скобка, помещается всегда в первой позиции перфокарты; N — номер строки в книге, после которой будут вставляться новые строки ($1 \leqslant N \leqslant 9999$).

Вставляемые строки располагаются вслед за строкой с номером N в любом количестве, и конец вставки опознается появлением другого оператора конкретизации.

Оператор удаления имеет формат

)DEL $N_1[,N_2]$

где N_1 — первая из удаляемых строк, а N_2 — последняя. Если удаляется одна строка, то N_2 не пишется. Как и в операторе добавления, $1 \leqslant N_1 < N_2 \leqslant 9999$.

Оператор замены имеет формат

)REP $N_1[,N_2]$

где $1 \leqslant N_1 < N_2 \leqslant 9999$. Вставляемые строки в любом (в том числе большем и меньшем) количестве помещаются во входном потоке вслед за оператором замены, как это делается при добавлении.

Оператор перенумерации имеет формат

)PRL N

После этого оператора помещаются перфокарты с пустыми 1—72 колонками и новыми номерами в колонках 73—76, которые будут присвоены операторам, начиная с N-го.

Последний из операторов конкретизации носит название *оператора конца* и имеет формат

)END[V.M[,C]]

Присутствие его среди операторов конкретизации после каждого оператора UPDATE обязательно, причем он должен быть последним среди этих операторов.

Значения V.M, задаваемые в операторе конца, могут не совпадать с задаваемыми в операторе UPDATE: первые присваиваются новой версии книги, а вторые используются при ее опознании. Наличие в операторе конца параметра C обеспечивает автоматическое наращивание кода V.M при последующих обновлениях, а отсутствие блокирует это наращивание.

Операторы конкретизации для каждого оператора UPDATE должны быть расположены в порядке возрастания значений N, N_1, N_2, ..., а если этого не удается достичь, то нужное преобразование исходного модуля осуществляется с помощью двух и более операторов UPDATE.

Пример 34. В книге F.D17 (версия 0, модификация 0) подбиблиотеки ФОРТРАНа удалить строки с 15 по 23, после 30-й строки добавить десять новых, а строки с 6 по 10 заменить новым текстом программы. Старую редакцию книги сохранить под именем F.D18.

Решение.

```
// JOB ПРИМ34
// EXEC MAINT
   UPDATE F.D17, F.D18,0.0
) REP 6,10
   <перфокарты трех новых операторов>
) DEL 15,23
) ADD 30
   <перфокарты десяти добавляемых операторов>
) END 0.0,C
/*
/&
```

Пример 35. На дисковом устройстве с адресом X'13A' расположен пакет дисков с инвентарным номером 000437, в котором в цилиндрах 10—59 размещена личная библиотека исходных модулей.

Из модуля P.ABCD7 (версия 0, модификация 7) этой библиотеки удалить операторы 25—35. Старую редакцию модуля не сохранять.

Решение.

```
// JOB ПРИМ35
// DLBL IJSYSSL
* УКАЗЫВАЕТСЯ ИМЯ ФАЙЛА SL
// EXTENT SYSSLB, 000437,1,7,100,500
* УКАЗАНЫ ЛУВВ, НОМЕР ПАКЕТА, ТИП
* УЧАСТКА, ЕГО НОМЕР И АДРЕСНЫЕ ДАННЫЕ
// ASSGN SYSSLB,X'13A'
* ДЛЯ ЛИЧНОЙ SL НАЗНАЧАЕТСЯ ЛУВВ
// EXEC MAINT
   UPDATE P.ABCD7,0.7
) DEL 25,35
) END 0.7,C
/*
/&
```

При наличии ошибок в операторе UPDATE и операторах конкретизации БИБЛИОТЕКАРЬ может обновление не выполнить. В частности:

— если в UPDATE имена книг неверны или наименование книги нарушает принцип уникальности, то обновление не производится, а последующие операторы контролируются на правильность и программист получает сообщения по возможности обо всех ошибках;

— аналогично БИБЛИОТЕКАРЬ реагирует на неправильное указание в UPDATE номера версии и модификации книги;

— при неверном значении nn обновление выполняется как при nn=1;

— если операторы конкретизации записаны неверно или нарушен порядок следования номеров обрабатываемых строк, БИБЛИОТЕКАРЬ действует как указано выше.

— наличие параметра N_2 в операторах добавления и перенумерации игнорируется.

5.4.5. Сжатие. Выше уже указывалось, что при удалении элементов библиотек удаляются только их записи в оглавлениях, тогда как сами удаляемые модули продолжают занимать место в памяти. Аналогичное положение имеет место и при обновлении. Все это оправдано, так как перемещение всех модулей на новые места при каждом удалении или обновлении вызывает большие затраты времени процессора. Но из соображений экономии памяти необходимо иметь операцию реорганизации библиотек, физически устраняющую всю став-

шую ненужной информацию. Такой операцией для БИБЛИОТЕКАРЯ ДОС/ЕС и является сжатие. Операция сжатия существует в двух вариантах; сжатие, выполняемое по требованию программиста, и автоматическое сжатие.

Оператор сжатия в первом вариаите имеет формат

CONDS <список библиотек>

причем список библиотек может состоять как из имени одной библиотеки или полного их набора, так и из любого частичного набора, например:

CONDS RL
CONDS CL, SL
CONDS CL, RL, SL

Для сжатия личных библиотек требуется наличие назначений для этих библиотек; при этом соответствующие системные библиотеки не сжимаются.

Пример 36. Удалить из библиотеки CL транслятор с ФОРТРАНа и сжать ее. Сжать также личную библиотеку SL, данные о которой приведены в примере 35.

Решение:

```
// JOB ПРИМ36
// DLBL IJSYSSL
// EXTENT SYSSLB,000437,1,0,100,500
// ASSGN SYSSLB,X'13A'
// EXEC MAINT
   DELETC FORT.ALL
   CONDS CL,SL
/*
/&
```

Оператор автоматического сжатия имеет формат

CONDL <список условий>

в котором

<список условий> : : =<имя библиотеки>= =nnnnn|<имя библиотеки>=nnnnn,<список условий>

<имя библиотеки> : : =CL|RL|SL

где nnnnn — не более чем пятизначное целое число.

Если в качестве nnnnn взят нуль, то сжатие выполняться не будет. Наоборот, если число nnnnn больше, чем число блоков, выделеных для обрабатываемой библиотеки, то сжатие будет производиться при каждом

обращении к этой библиотеке. В общем случае сжатие производится, если количество свободных блоков библиотеки при очередном обращении к ней окажется меньшим, чем число ппппп.

Пример 37. Личная библиотека RL имеет те же параметры, что и в примере 36, а личная библиотека SL расположена на том же пакете дисков с 60-го по 189-й цилиндры включительно. Установить параметры автоматического сжатия ппппп=20, 50 и 50 блоков для системных библиотек CL, RL и SL соответственно и ппппп=100 и 100 блоков для личных библиотек.

Решение.

```
// JOB ПРИМ37
// EXEC MAINT
   CONDL CL=00020,RL=00050,SL=00050
/*
// DLBL IJSYSSL
// EXTENT SYSSLB,000437,1,0,100,500
// ASSGN SYSSLB,X'13A'
// DLBL IJSYSRL
// EXTENT SYSRLB,000437,1,0,600,1300
// ASSGN SYSRLB,X'13A'
   CONDL RL=00100,SL=00100
/*
/&
```

5.4.6. Распределение. Входящая в состав БИБЛИОТЕКАРЯ функция *распределения* позволяет перераспределять количество цилиндров и дорожек резидентного файла между системными библиотеками и их оглавлениями. Для личных библиотек функция распределения не применяется. Запрещено также применять эту функцию в период активности разделов переднего плана (т. е. когда в F1 или F2 идет пакетная обработка или выполняются одиночные программы), а также во время работы программы ВНИМАНИЕ.

Функция распределения позволяет:

— изменять размеры системных библиотек и их оглавлений;

— исключать системные библиотеки RL и SL;

— добавлять, если их не было, пустые системные библиотеки RL и SL.

При этом автоматически происходит сжатие библиотек.

Оператор распределения имеет формат

ALLOC <список распределений>

где

<список распределений> : : = <распределение> | <распределение>,<список распределений>

причем количество распределений не должно превышать трех.

Формат распределения определяется правилами

<распределение> : : = <имя библиотеки> = <количество цилиндров> (<количество дорожек>)
<имя библиотеки> : : =CL|RL|SL
<количество цилиндров> : :=nnn
<количество дорожек> : : =nn

nnn и nn — целые числа, соответственно не более чем трех- и двухзначные.

Исключение библиотеки происходит при выделении ей нулевого количества цилиндров и дорожек, а создание — когда при перераспределении ранее отсутствующей библиотеки выделяются ненулевые значения nnn и nn. Следует помнить, что последний из цилиндров резидентного файла, выделяемый для библиотек, 197-й.

Для выполнения операции распределения операторами DLBL и EXTENT надо определить резидентный файл, используя его имя IJSYSRS, и идентификатор DOS/ES SYSTEM RESIDENCE FILE, указав его нижнюю границу: цилиндр 0, дорожка 1 и верхнюю, включающую цилиндр меток.

Пример 38. Резидентный файл, включая программу первоначальной загрузки (ППЗ) и цилиндр меток, занимает 180 цилиндров; он размещен на пакете № 123456 в устройстве с адресом X'101'. Выделить для библиотеки CL 120 цилиндров, а для его оглавления 12 дорожек; для RL количество цилиндров и дорожек соответственно 40 и 4, а для SL—10 и 3. Остальная часть пакета, кроме цилиндра меток, выделяется под область пользователя.

Решение.

```
// JOB ПРИМ38
// DLBL IJSYSRS,'DOS/ES SYSTEM RESIDENCE FILE',,DA
* ПРЕДПОЛАГАЕТСЯ ПРЯМАЯ ОРГАНИЗАЦИЯ ФАЙЛА
// EXTENT SYSRES,123456,1,0,1,1799
* ИСКЛЮЧЕНА ДОРОЖКА, ЗАНЯТАЯ ППЗ. ASSGN НЕ ДЕ-
* ЛАЕТСЯ, Т. К. ДЛЯ SYSRES ДЕЙСТВУЕТ ПОСТОЯННОЕ НА-
* ЗНАЧЕНИЕ.
// EXEC MAINT
   ALLOC CL=120(12),RL=40(4),SL=10(3)
/*
/&
```

5.5. Сервисные функции БИБЛИОТЕКАРЯ

К сервисным функциям относятся:

— печать библиотек, их элементов, оглавлений, а также состояния ДОС и личных библиотек;

— перфорация библиотек и их элементов;

— совместное выполнение печати и перфорации.

Программы, выполняющие сервисные функции, вызываются операторами вида

// EXEC CSERV для библиотеки абсолютных модулей,

// EXEC RSERV для библиотеки объектных модулей,

// EXEC SSERV для библиотеки исходных модулей,

// EXEC JSERV для перемещения модулей из библиотеки исходных модулей на ленту или диск в формате, необходимом для передачи трансляторам и, наконец,

// EXEC DSERV для печати оглавлений и (или) состояния библиотек ДОС/ЕС.

Вид сервисной функции определяется следующим после оператора EXEC оператором БИБЛИОТЕКАРЯ: DSPLY для функции печати, PUNCH для функции перфорации и DSPCH для функции печати и перфорации.

5.5.1. Печать. Оператор печати имеет формат

DSPLY <список>

где

<список> : : = <элемент списка> | <элемент списка>, <список>

При выводе на печать из библиотеки CL

<элемент списка> : : = <имя фазы> | <имя программы>.ALL | ALL

причем <имя программы> — первые четыре символа, общие в именах всех фаз программы.

Вид элемента списка определяет объем печати: печатается указанная фаза, вся программа или вся библиотека.

При выводе на печать из библиотеки RL

<элемент списка> : : = <имя модуля> | <имя программы>.ALL | ALL

и в отличие от CL <имя программы> — первые три символа, общие для всех модулей программы.

При выводе на печать из библиотеки SL

<элемент списка> : : = <символ подбиблиотеки>. <имя книги> | <символ подбиблиотеки>.ALL | ALL

Функция печати применяется к личным библиотекам, если для них имеются назначения, и к системным в противном случае.

При печати оглавлений

<список> : : = <список оглавлений> | ALL

<список оглавлений> : : = <имя оглавления> | <имя оглавления>,<список оглавлений>

<имя оглавления> : : = TD | CD | RD | SD

где TD — имя транзитного оглавления; CD — имя оглавления CL; RD — имя оглавления RL; SD — имя оглавления SL.

Для печати оглавлений с упорядочением их по алфавиту вместо оператора DSPLY применяется оператор DSPLYS.

Для выдачи на печать состояния системных (и личных, если они назначены) библиотек без печати их оглавлений употребляется упомянутый выше оператор

```
// EXEC DSERV
```

без операндов. Выдаваемый на печать материал библиотек представляет собой заголовок и текст.

Для библиотек CL заголовок состоит из имени фазы и ее длины в байтах, текст печатается в 16-ричном формате. Для библиотеки RL заголовок содержит имя модуля и количество блоков, а текст можно выдать в 16-ричном или символьном формате. Для библиотеки SL заголовок содержит полное имя книги (включая символ подбиблиотеки) и количество блоков, текст символьный с операторами BKEND до и после текста исходного модуля.

Пример 39. Из библиотеки исходных модулей удалить строки 33—38 модуля A.XABC14 и напечатать текст полученного модуля. Напечатать также оглавление этой библиотеки с упорядочением его по алфавиту.

Решение.

```
// JOB ПРИМ39
// EXEC MAINT
   UPDATE A.XABC14
```

```
) DEL 33,38
) END
/*
// EXEC SSERV
   DSPLY A.XABC14
/*
// EXEC DSERV
   DSPLYS SD
/*
/&
```

Пример 40. Переименовать фазу A1B2C3 в M1H1P1, напечатать ее на АЦПУ, после чего напечатать состояние ДОС.

Решение.

```
// JOB ПРИМ40
// EXEC MAINT
   RENAMC A1B2C3, M1H1P1
/*
// EXEC CSERV
   DSPLY M1H1P1
/*
// EXEC DSERV
/&
```

5.5.2. Перфорация. Перфорация библиотек и их элементов производится как для создания страховочных колод, так и для последующих работ с выдаваемым материалом: трансляции, редактирования, каталогизации. Возможна и фиктивная перфорация, когда в качестве SYSPCH выбран накопитель на магнитной ленте или диске.

Объектами перфорации могут быть как отдельные модули, так и программы (в SL подбиблиотеки) и библиотеки в целом. Отперфорированный модуль (фаза) из библиотеки CL состоит из карт текста (TXT) и карт ESD и END, которые в фазе отсутствуют и создаются БИБЛИОТЕКАРЕМ на основании анализа записи фазы в оглавлении CL. Поэтому выдаваемые на перфорацию фазы могут быть использованы РЕДАКТОРОМ для каталогизации. Для этого перед колодой карт фазы БИБЛИОТЕКАРЬ строит и выдает карту PHASE. Карты выдаются в таком формате, что при их использовании перемещений фазы в памяти не допускается.

Таким образом, в состав колоды выдаваемой фазы входят:

— карта PHASE с именем фазы и адресом загрузки;

— карта ESD с элементом SD, в котором повторяется имя фазы и адрес загрузки, а также дается длина фазы;

— карты TXT в необходимом количестве, каждая из которых содержит адрес первого байта, количество байтов (обычно 56) и их содержание;

— карта END с адресом входа в фазу и адресом ее конца.

После карт фазы перфорируется оператор /*, а на SYSLST печатается список отперфорированных фаз. В колонках 77—80 перфорируется номер карты, начиная с 0001 для каждой выдаваемой фазы.

Модули, выдаваемые на перфорацию из библиотеки RL, начинаются с карты с оператором CATALR и именем модуля и также оканчиваются картой с оператором /*. Нумерация карт и печать на SYSLST производится, как и для фаз библиотеки CL.

При перфорации из библиотеки SL перед картами выдаваемой книги перфорируется карта с оператором CATALS и именем книги, далее идет карта BKEND, а после карт книги — заключительная карта BKEND и карта с оператором /*. Нумерация и печать на SYSLST не отличается от описанных для CL и RL.

Для перфорации программист пользуется оператором БИБЛИОТЕКАРЯ, имеющим формат

PUNCH <список объектов>

а для перфорации из SL

PUNCH <список объектов> [,CMPRSD]

причем последний операнд означает, что программист хочет выдать информацию на перфокарты в сжатом формате.

Определение списка объектов перфорации:

<список объектов> : : = <элемент списка> | <элемент списка>, <список объектов> | ALL

Для CL

<элемент списка> : : = <имя фазы> | <имя программы>.ALL

Для RL < имя фазы> заменяется на <имя модуля>, а для SL — на <символ подбиблиотеки>.<имя книги>, <имя программы>.ALL на <символ подбиблиотеки>.ALL.

Пример 41. Создать страховочную колоду перфокарт для транслятора с ФОРТРАНа, модуль Р. АВС123 из библиотеки исходных модулей оттранслировать и также выдать на перфокарты. В качестве SYSPCH и SYSIPT использовать лентопротяжное устройство с адресом X'284'.

Решение.

```
// JOB ПРИМ41
// EXEC CSERV
   PUNCH FORT.ALL
/*
// ASSGN SYSPCH,X'284'
// MTC REW,X'284'
// MTC WTM,SYSPCH
* ЭТИМИ ОПЕРАТОРАМИ МЫ НАЗНАЧИЛИ SYSPCH, ПЕРЕ-
* МОТАЛИ ЛЕНТУ НА НАЧАЛО И ПОСТАВИЛИ МАРКЕР.
// EXEC JSERV
   PUNCH P.ABC123
/*
* ТЕПЕРЬ P.ABC123 ПЕРЕНЕСЕН НА МАГНИТНУЮ ЛЕНТУ
* БЕЗ КАКИХ-ЛИБО НЕЖЕЛАТЕЛЬНЫХ ДЛЯ ТРАНСЛЯЦИИ
* ДОБАВЛЕНИЙ.
// OPTION DECK
// RESET SYSPCH
* ДЛЯ СИСТЕМНОГО ПЕРФОРАТОРА ВОССТАНОВЛЕНО ПО-
* СТОЯННОЕ НАЗНАЧЕНИЕ.
// ASSGN SYSIPT,X'284'
// MTC REW,X'284'
// EXEC PL/I
/&
```

5.5.3. Печать и перфорация. Объединение функций печати и перфорации позволяет значительно сэкономить машинное время по сравнению с раздельным выполнением этих функций. Возможности этого вида обслуживания совпадают с возможностями перфорации, причем форматы выдачи на печать и на перфокарты совпадают с теми, которые были описаны в пп. 5.5.1, 5.5.2.

Формат оператора БИБЛИОТЕКАРЯ, обеспечивающего печать и перфорацию, имеет вид

DSPCH <список объектов>

Ниже приводится ряд примеров использования описанных в § 5.5 функций БИБЛИОТЕКАРЯ.

Пример 42. Выдать на печать исходный модуль A.B17366 из системной библиотеки SL, отперфорировать страховочную колоду транслятора с КОБОЛа и напечатать оглавления всех библиотек.

Решение.

```
// JOB ПРИМ42
// EXEC SSERV
   DSPLY A.B17366
/*
```

```
// EXEC CSERV
   PUNCH FCOB.ALL
/*
* FCOBOL — ИМЯ ПЕРВОЙ ФАЗЫ ТРАНСЛЯТОРА С КОБОЛА.
* FCOB — ПЕРВЫЕ 4 СИМВОЛА, НЕОБХОДИМЫЕ ДЛЯ УКА-
* ЗАНИЯ ОБ УДАЛЕНИИ ВСЕХ ФАЗ ТРАНСЛЯТОРА.
// EXEC DSERV
   DSPLY ALL
/*
/&
```

Пример 43. Выдать на печать оглавление личной библиотеки SL, расположенной на цилиндрах 30—89 дискового пакета 000437, установленного на устройство с адресом X'107'.

Решение.

```
// JOB ПРИМ43
// DLBL IJSYSSL
// EXTENT SYSSLB,000437,1,0,300,600
// ASSGN SYSSLB,X'107'
// EXEC DSERV
  DSPLY SD
/*
/&
```

5.6. Создание и копирование

Функция *создания* нового резидентного файла на основе существующего и новых личных библиотек предназначена для частичного копирования исходных материалов, тогда как функция *копирования* — для получения идентичных копий. При копировании резидентного файла допускается лишь попутное перераспределение памяти между библиотеками.

5.6.1. Создание и копирование резидентного файла. Функции создания и копирования резидентного файла обеспечиваются программой CORGZ, которая за одно обращение к ней позволяет получить один резидентный файл.

На новый резидентный пакет копируются:

— две записи программы первоначальной загрузки;

— СУПЕРВИЗОР, РЕДАКТОР, УПРАВЛЕНИЕ ЗАДАНИЯМИ;

— транзиты СУПЕРВИЗОРА;

— цилиндр меток, содержащий стандартные метки ДОС/ЕС и разделов (при мультипрограммном режиме);

— библиотеки или их отдельные элементы.

Область пользователя исходного резидентного пакета не копируется.

При полном копировании получается страховочный пакет, при частичном — новый, но сокращение библиотек не должно затрагивать таких составных частей CL, как СУПЕРВИЗОР, РЕДАКТОР, УПРАВЛЕНИЕ ЗАДАНИЯМИ, транзиты, без которых система неработоспособна. Поэтому даже при отсутствии дополнительных операторов одним обращением

// EXEC CORGZ

обеспечивается копирование всех перечисленных составляющих ДОС/ЕС. Для копирования библиотек применяются операторы из серии COPY: COPYC для CL, COPYR для RL и COPYS для SL. Для копирования всех библиотек применяется оператор COPY. Формат этих операторов таков:

<оператор> : : = <код оператора> <список объектов>

<код оператора> : : =COPYC|COPYR|COPYS|COPY

<список объектов> : : =<объект>|<объект>, <список объектов>|ALL

В зависимости от библиотеки

<объект> : : = <фаза>|<модуль>|<книга>| <программа>.ALL|<подбиблиотека>.ALL

Оператор COPY используется только в виде COPY ALL. Перераспределение памяти между библиотеками в новом или страховочном пакете осуществляется оператором

ALLOC<список распределений>

который в отличие от имеющего такое же написание и такой же список распределений оператора, описанного в п. 5.4.6, воздействует на распределение памяти между библиотеками в создаваемом пакете, не затрагивая исходный.

Среди операторов БИБЛИОТЕКАРЯ, необходимых программе CORGZ, оператор ALLOC, если он применяется, должен быть первым.

Для создания нового пакета необходимо логическое устройство SYS002 (ему необходимо назначить одно из дисковых физических устройств). На нем, как и при использовании функции распределения (см. п. 5.4.6), определяется резидентный файл IJSYSRS с идентификатором 'DOS/ES SYSTEM RESIDENCE FILE', нижней

границей 1 (дорожка 1 нулевого цилиндра) и верхней 1799.

Пример 44. Скопировать существующий файл на пакет дисков № 100999, установленный на устройстве с адресом X'105'.

Решение.

```
// JOB ПРИМ44
// DLBL IJSYSRS,DOS/ES RESIDENCE FILE',,DA
// DLBL IJSYS02,,,DA
// EXTENT SYS002,100999,1,0,1979
// ASSGN SYS002,X'105'
// EXEC CORGZ
   COPY ALL
/*
/&
```

Пример 45. Создать новый резидентный пакет, удалив из него библиотеки RL и SL. Остальные условия взять из примера 44.

Решение.

```
// JOB ПРИМ45
// DLBL    )
// EXTENT  } как в примере 44
// ASSGN   )
// EXEC CORGZ
   ALLOC CL=170(20),RL=0(0),SL=0(0)
   COPYC ALL
/*
/&
```

5.6.2. Создание и копирование личных библиотек. При создании и копировании личных библиотек исходными могут быть и системные и личные библиотеки. В обоих случаях для создания этих библиотек используется тот же оператор

// EXEC CORGZ

что и при копировании резидентного файла. Аналогично создание новой библиотеки состоит из распределения памяти и частичного или полного копирования библиотеки.

Для новой библиотеки CL, создаваемой на SYSCLB, определяется файл IJSYSCL, для RL, создаваемой на SYSRLB, — файл IJSYSRL, для SL на SYSSLB — файл IJSYSSL. При назначении областей диска для всех личных библиотек может быть выбрано одно общее дисковое устройство, но участки в операторах EXTENT должны быть выделены разные. Для CYSCLB назначение должно быть постоянным.

Для создания личных библиотек на основе системных используется оператор БИБЛИОТЕКАРЯ вида

NEWVOL <список распределений>

играющий здесь роль оператора ALLOC. Объем копирования определяется в операторах COPYC, COPYR и COPYS, как это описано в п. 5.6.1. В списке может быть одно, два или три распределения:

<распределение> : : = <идентификатор библиотеки> = <количество цилиндров> (<количество дорожек>)

<идентификатор библиотеки> : : =CL|RL|SL.

Оператор NEWVOL, применяемый для создания и копирования библиотек, необходим в любом случае. Он пишется первым, а за ним следуют необходимые операторы COPYC, COPYR или COPYS. С помощью одного оператора

// EXEC CORGZ

можно создать не более чем одну библиотеку каждого вида, а также две или три разноименные библиотеки.

При выделении с помощью оператора NEWVOL памяти для какой-либо библиотеки и пропуске в дальнейшем тексте задания соответствующего оператора COPY создается «пустая» библиотека, которую затем программист заполняет по своему усмотрению.

Пример 46. На дисковом пакете № 010101, установленном на устройстве с адресом X'123', создать копию системной бнблиотеки RL и пустую библиотеку SL, выделив для первой 100 цилиндров, а для второй 50 и по 10 дорожек для оглавления в каждой.

Решение.

```
// JOB ПРИМ46
// DLBL IJSYSRL,,,DA
// EXTENT SYSRLB,010101,1,0,0,1000
// ASSGN SYSRLB,X'123'
// DLBL IJSYSSL,,,DA
// EXTENT SYSSLB,010101,1,0,1000,500
// ASSGN SYSSLB,X'123'
// EXEC CORGZ
   NEWVOL RL=100(10),SL=50(10)
   COPYR ALL
/*
/&
```

При создании личных библиотек на основе уже имеющихся личных библиотек вместо NEWVOL используется оператор PRVCPRV, также играющий роль оператора ALLOC и помещаемый на первом месте. Исходные биб-

лиотеки располагаются на устройствах SYS000 (CL), SYS004 (RL) и SYS003 (SL) и для них определяются имена файлов PRCLALT, PRRLALT и PRSLALT соответственно. Создаваемым библиотекам присваиваются имена файлов IJSYSCL, IJSYSRL и IJSYSSL, они располагаются на устройствах SYSCLB, SYSRLB и SYSSLB.

В использовании операторов COPYC, COPYR и COPYS отличий от рассмотренного выше случая нет.

Пример 47. На пакете дисков № 010101 имеются личная библиотека RL, занимающая первые 100 цилиндров, и личная библиотека SL (последующие 50 цилиндров). Количество дорожек для оглавлений по 10. Требуется создать новые библиотеки на пакетах № 000123 и 000124, выделив для каждой по пакету и разместив их на первых 170 цилиндрах (свободны дисковые устройства с адресами X'107'; X'108' и X'109').

Решение.

```
// JOB ПРИМ47
// DLBL PRRLALT,,,DA
// EXTENT SYS004,010101,1,0,0,1000
// ASSGN SYS004,X'107'
// DLBL PRSLAT,,,DA
// EXTENT SYS003,010101,1,0,1000,500
// ASSGN SYS003,X'107'
// DLBL IJSYSRL,,,DA
// EXTENT SYSRLB,000123,1,0,0,1700
// ASSGN SYSRLB,X'108'
// DLBL IJSYSSL,,,DA
// EXTENT SYSSLB,000124,1,0,0,1700
// ASSGN SYSSLB,X'109'
// EXEC CORGN
   PRVCRV RL=170(20),SL=170(20)
   COPYR ALL
   COPYC ALL
/*
/&
```

Наличие нескольких личных библиотек одинакового типа может вызвать у программиста некоторые затруднения, для преодоления которых в разных шагах задания вводятся различные пары операторов DLBL и EXTENT, а если библиотеки расположены на разных устройствах, то и различные операторы ASSGN.

5.6.3. Слияние библиотек. В ДОС/ЕС, начиная с версии 2.0, с помощью оператора MERGE вводится функция *слияния*, т. е. пересылки в уже существующую библиотеку (системную или личную) модулей, хранящихся в другой (также системной или личной) библиотеке. Возможно слияние библиотек, находящихся на двух различных резидентных пакетах.

Оператор слияния имеет формат

MERGE <идентификатор 1>,<идентификатор 2>

причем MERGE пишется после хотя бы одного пробела, а его операнды означают:

<идентификатор 1>::=<идентификатор файла, откуда идет копирование>

<идентификатор 2>::=<идентификатор файла, на который идет копирование>

Значения операнда <идентификатор 1>:

RES при копировании с резидентного файла;

PRV при копировании из личной библиотеки, причем файл PRV устанавливается на логическом УВВ SYS000 для копирования библиотеки исходных модулей, на SYS001 для объектных и на SYS003 для абсолютных;

NRS при копировании с резидентного файла, который в данный момент не играет роли файла, управляющего работой машины. В этом случае файл NRS ставится на логическое УВВ SYS002.

Значения операнда <идентификатор 2>:

NRS при копировании в новый или любой другой резидентный файл, но не тот, который управляет работой машины (в этом случае NRS также ставится на УВВ SYS002);

RES при копировании в действующий резидентный файл;

PRV при копировании в одну из личных библиотек (но в этом случае личная библиотека, в которую идет копирование, соответственно должна находиться на SYSCLB, SYSRLB или на SYSSLB).

Пример 48. Скопировать транслятор ПЛ/1 с заимствованного резидентного файла на свой. Использовать УВВ с адресом X'194'.

Решение.

```
// JOB ПРИМ48
// ASSGN SYS002,X'194'
// EXEC CORGZ
   MERGE NRS,RES
   COPYC PL/I
/*
/&
```

Пример 49. Внести из личной библиотеки «А.ИВАНОВ» в личную библиотеку «Б.ПЕТРОВ» исходный модуль А.ААВВСС. Библиотеки расположены:

— «А.ИВАНОВ» на дисковом пакете № 246810, на цилиндрах 30—69,

— «Б.ПЕТРОВ» — на пакете № 222222, на цилиндрах 80—119. Использовать физические дисковые УВВ с адресами X'195' и X'196

Решение.

```
// JOB ПРИМ49
// ASSGN SYS000,X'195'
// ASSGN SYSSLB,X'196'
// DLBL ABC12,'А.ИВАНОВ'
// EXTENT SYS000,246810,1,0,300,400
// DLBL IJSYSSL'Б.ПЕТРОВ'
// EXTENT SYSSLB,222222,1,0,800,400
// EXEC CORGZ
   MERGE PRV, PRV
   COPYS A.AABB.CC
/*
/&
```

БИБЛИОТЕКАРЬ ДОС/ЕС версии 2.0 позволяет программисту перенести на другой дисковый пакет также фазу, представляющую программу ПЕРВОНАЧАЛЬНАЯ ЗАГРУЗКА, записанную на резидентном пакете вне библиотек. Для этого после оператора MERGE ставится оператор

```
COPYI   $$A$  IPL2
```

6. Редактор

6.1. Процесс редактирования

Разбиение процесса преобразования программы от текста на языке программирования к машинным кодам на две стадии — трансляцию и редактирование открывает возможности создания программ по частям — модулям, допускающим независимую трансляцию. Полученные в результате трансляции объектные модули строятся трансляторами ДОС/ЕС в таком формате, что абсолютные адреса команд, переменных и обращений к другим модулям не вырабатываются, а к модулю придается информация, позволяющая выполнить эту работу на этапе редактирования.

Основные задачи РЕДАКТОРА ДОС/ЕС:

— собирать объектные модули или составляющие их программные секции в единую программу;

— перерабатывать тексты объединяемых модулей так, чтобы вырабатываемая программа могла работать будучи загруженной в определенное место памяти;

— помещать полностью готовую к использованию

программу на хранение в библиотеку абсолютных модулей.

В ранних версиях ДОС/ЕС РЕДАКТОР мог работать только как программа фонового раздела памяти. В более поздних версиях, начиная с версии 2.0, РЕДАКТОР, будучи самоперемещающейся программой, может работать как в разделе BG, так и в разделах переднего плана.

Как и программа УПРАВЛЕНИЕ ЗАДАНИЯМИ, РЕДАКТОР может предоставлять программисту некоторый комплекс услуг, именуемый режимом. Выбор того или иного режима программист осуществляет с помощью одного из управляющих операторов РЕДАКТОРА. С помощью других операторов программист может управлять модульной структурой программных фаз, адресами их загрузки в оперативную память и назначать точку входа в программу, если ее необходимо иметь не в начале первой фазы, а в каком-либо другом ее месте.

Задаваемые программистом на исходном языке связи между модулями, в том числе между модулями программиста и системными модулями, РЕДАКТОР преобразует в абсолютные адреса, вырабатывая *программную фазу*. Структура программ, состоящих из многих фаз, планируется программистом на исходном языке, но не все языки программирования имеют для этого достаточно выразительные средства. Так, многофазные программы в ДОС/ЕС можно строить с помощью ПЛ/1 и языка АССЕМБЛЕРА. В такой программе на исходном языке планируется последовательность вызова фаз, способы передачи управления очередной фазе и использование оперативной памяти.

Редактируя программу, состоящую из многих фаз, РЕДАКТОР применяет специально для этого предназначенные средства.

Фаза программы, подготовленная РЕДАКТОРОМ, содержит только текст программы в абсолютных адресах. Объектные же модули, поступающие на обработку к РЕДАКТОРУ, содержат кроме текста, относящегося к операторам программы, дополнительную информацию, позволяющую настраивать их на любое место памяти и связывать с другими модулями.

Процесс редактирования можно представить в виде двух этапов: формирования текста фаз и формирования записей оглавления.

Единицами входной информации РЕДАКТОРА являются объектные модули и программные секции — некоторые относительно самостоятельные части модуля, например, подпрограммы или участки памяти, выделяемые под общие области.

Функции РЕДАКТОРА показаны на рис. 6.1

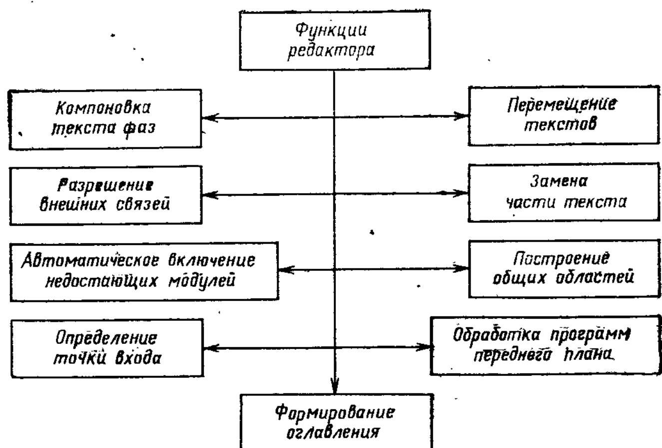


Рис. 6.1. Функции РЕДАКТОРА ДОС/ЕС.

Компоновка текстов фаз производится из текстов объектных модулей, задаваемых программистом, в том порядке, в каком они указаны в его задании. Наряду с модулями, включаемыми целиком, в программу могут входить также лишь некоторые из секций, составляющих модуль. При создании текста фазы вычисляются абсолютные адреса, а внешние имена используются для установления связей между модулями. РЕДАКТОР производит разрешение связей между модулями не только за счет присоединения к фазе модулей, указанных программистом, но и с помощью присоединения к ней модулей из системных и личных библиотек. Однако программист имеет возможность ограничить действия РЕДАКТОРА только перечнем явно присоединяемых модулей.

Часть текста модулей при редактировании может обновляться, аналогичная операция производится БИБЛИОТЕКАРЕМ над исходными модулями.

После редактирования для каждой фазы определяется точка входа в нее. Специальную обработку проходят программы, предназначенные для работы в разделах переднего плана. Общие области, необходимые для обмена информацией между фазами, РЕДАКТОР строит на основании текстов объектных модулей, причем области, имеющие в модулях одинаковые имена, объединяются в одну; длина этой области выбирается равной максимальной из длин, указанных в обьектных модулях. Для разных имен строятся разные области и располагаются они в области памяти, доступной использующим ее фазам.

При формировании оглавления программы строятся записи для каждой из фаз. В такой записи содержатся: имя фазы, адрес ее загрузки, точка входа и адрес фазы на диске. Совокупность таких записей — *оглавление фаз программы* — размещается в системной рабочей области.

Использование оглавления зависит от способа использования обрабатываемой программы. В режиме LINK вырабатываемые фазы должны выполняться сразу после редактирования с помощью оператора

// EXEC

Оглавление в этом случае остается в системной области и не переносится в CL а программист должен проследить, чтобы между оператором редактирования

// EXEC LNKEDT

и оператором

// EXEC

не встретились операторы

// OPTION LINK
// OPTION CATAL

которые могут разрушить созданную программу.

Так как в многофазной программе фаза, вызванная первой, вызывает последующие по именам, то в режиме LINK допускается создание и выполнение только одно-

фазных программ или программ, использующих ранее каталогизированные в CL фазы.

При многократном использовании программы она каталогизируется в библиотеку абсолютных модулей, и это единственная библиотечная функция, выполняемая не БИБЛИОТЕКАРЕМ, а РЕДАКТОРОМ.

При каталогизации в библиотеку CL применяется режим CATAL, что приводит к размещению создаваемых фаз в CL и переносу их оглавления из системной области в оглавление CL. Выполнение каталогизированной в CL программы происходит после считывания с SYSRDR оператора

// EXEC<имя первой фазы>

В процессе редактирования кроме программных фаз и их оглавления РЕДАКТОР вырабатывает некоторую побочную информацию, осуществляя обзор созданных фаз, позволяющий установить структуру полученной программы, и посылая программисту диагностические сообщения, позволяющие обнаружить ошибки и неточности в задании операторов РЕДАКТОРА.

Скорость редактирования повышается, если в распоряжении РЕДАКТОРА имеется не менее 14К байтов памяти.

Имена используемых РЕДАКТОРОМ логических УВВ и располагающихся на них файлов приводятся в табл. 6.1. Используются эти данные при формировании операторов назначения физических УВВ этим логическим устройствам.

Описанный выше путь прохождения информации в процессе редактирования показан на рис. 6.2.

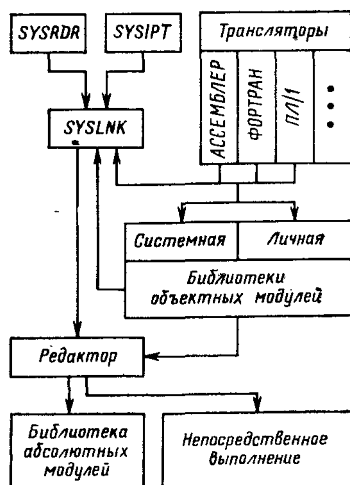


Рис. 6.2. Движение информации в процессе редактирования программы в ДОС/ЕС.

Таблица 6.1

Имя УВВ	Способ использования	Имя файла	Тип физического УВВ
SYSRES	Резиденция ДОС/ЕС	IJSYSRS	Диски
SYSLNK	Входная информация РЕДАКТОРА	IJSYSLN	»
SYSRDR	Операторы РЕДАКТОРА и программы УПРАВЛЕНИЯ ЗАДАНИЯМИ	IJSYSIN	Диски, лента, перфокарточный ввод
SYSIPT	Чтение УПРАВЛЕНИЕМ ЗАДАНИЯМИ входной информации РЕДАКТОРА и перепись ее на SYSLNK	»	То же
SYSRLB	Личная RL	IJSYSRL	Диски
SYSLST	Вывод обзора фаз и диагностических сообщений	IJSYSLS	АЦПУ, диски, лента
SYSLOG	Связь с оператором	Нет	Пишущая машинка (в аварийном режиме — перфокарточный ввод и АЦПУ)
SYS001	Рабочий файл РЕДАКТОРА	IJSYS01	Диски, лента
SYSSLB	Личная SL	IJSYSSL	Диски

6.2. Структура объектных модулей

Объектные модули в ДОС/ЕС делятся на *стандартные* и *нестандартные*.

Трансляторы формируют обычно только стандартные модули и лишь в отдельных случаях могут перед модулем помещать операторы РЕДАКТОРА или БИБЛИОТЕКАРЯ. Модули, независимо от места их хранения, состоят из порций информации, имитирующих их размещение на перфокартах. Поэтому ниже будем говорить о перфокартах, из которых они состоят. Перфокарты стандартных модулей можно разделить на следующие пять типов:

1. Карты типа SYM, предназначенные для записи таблицы символических имен, использованных программистом в исходном модуле. Эти карты применяются при выдаче модуля на печать и РЕДАКТОРОМ игнорируются.

2. Карты типа ESD содержат данные, необходимые РЕДАКТОРУ для разрешения внешних связей модуля

с другими модулями, включаемыми в программу. На картах ESD имеются «элементы» по 16 байтов типа SD, PC, LD, ER и CM.

Элемент SD строится для символических имен именованных программных секций, т. е. частей объектного модуля, имеющих свои имена (это понятие, используемое в языке АССЕМБЛЕРА; в ФОРТРАНе и ПЛ/1 аналогичный объект — именованная внешняя область и переменная с описателем EXTERNAL).

Элемент SD содержит: имя секции, ее длину в байтах и адрес начала секции относительно начала модуля. Для каждого модуля транслятор создает не менее одного элемента SD (когда понятия секции и модуля совпадают).

Элемент PC строится АССЕМБЛЕРОМ для определения создаваемых им неименованных программных секций. Этот элемент содержит пробелы вместо имени секции, ее длину в байтах и адрес начала секции.

Элемент LD строится для *входного имени*, т. е. имени, определенного в данном модуле, а используемого в других модулях в качестве точки входа, имени данных и др. Он содержит символическое имя, его адрес в модуле, номер или имя секции, где оно определено.

Элемент ER строится для *внешнего имени*, т. е. для имени, используемого в данном модуле, но не определенного в нем. Для такого имени РЕДАКТОР ищет модуль, в котором было бы его определение, т. е. элемент LD этого имени.

Элемент CM является определением общей области для двух и более секций или подпрограмм и содержит имя области и ее длину в байтах.

3. Карты типа TXT содержат основной текст программы — команды и данные, а также счетчик байтов, адрес первого байта карты и номер программной секции, которой принадлежит текст. Эта информация используется РЕДАКТОРОМ при определении в тексте модуля абсолютных адресов.

4. Карты типа RLD содержат данные о частях текста, изменяемых при перемещениях, и необходимые для перемещения данные. Они составляют словарь перемещений.

5. Карта типа END является последней в модуле. Она может содержать символическое имя входной точки и длину модуля или секции в байтах.

Карты других типов добавляются в модуль обычно программистом. Так, для изменения текста модуля используются карты типа REP. Каждая из таких карт содержит адрес первого из изменяемых байтов, номер секции, в которой делаются изменения и новые коды для каждого из изменяемых байтов.

Количество изменяемых байтов на одной карте типа REP может быть в пределах 2—22.

Нестандартные модули делятся на *вызывающие* и *смешанные*.

Вызывающие модули состоят из управляющих операторов РЕДАКТОРА и любых перфокарт с пробелом в первой позиции. Последней картой такого модуля должна быть уже описанная выше карта END. Для каталогизации в RL часто встречающихся комбинаций управляющих операторов РЕДАКТОРА используются вызывающие модули.

Смешанные модули состоят из карт вызывающего модуля, за которыми следуют карты стандартного модуля с картой END в конце. Используются смешанные модули для упрощения процесса редактирования.

6.3. Разрешение внешних связей между модулями

Внешним связям между модулями, определяемым на исходном языке, при редактировании необходимо присвоить абсолютные адреса. Для этого используются карты (они же словари) ESD (см. § 6.2) всех модулей. Для разрешения конкретной связи требуется совпадение имен и идентифицирующих номеров.

Каждому элементу словаря ESD, кроме элементов типа LD, на этапе трансляции присваивается очередной в модуле номер. Элементы же LD, наоборот, ссылаются на некоторое имя другого элемента этого же или некоторого другого модуля и номер его могут получить лишь на этапе редактирования. Словари ESD просматриваются модуль за модулем в порядке, заданном программистом. Порядок же имен внутри модуля зависит от его структуры.

Для каждого внешнего имени, упомянутого в данном модуле с помощью элемента ER словаря ESD, нужно найти объекты, определенные в других модулях с по-

мощью элементов SD, LD и CM. Отыскание этих элементов позволяет найти адрес передачи управления из данной точки модуля за его пределы и определить, откуда взять недостающие данные или куда послать полученные результаты. В частности, по элементам ER находят необходимые для разрешения внешних связей библиотечные модули СУВВ, входные имена которых начинаются с символов IJ.

В любом модуле имеется один или больше элементов SD, определяющих именованную программную секцию или подпрограмму. Если такой элемент только один, он определяет адрес входа в модуль. При наличии же нескольких элементов SD можно осуществлять вход в разные точки модуля как из других модулей, так и из разных мест этого же модуля. Если внешнее имя какого-либо модуля, определенное одним из элементов ER его словаря ESD, совпадает с именем элемента SD в другом модуле, то РЕДАКТОР строит передачу управления в точку фазы, заданную этим SD (рис. 6.3), а при отсутствии такой точки входа в названных программистом модулях ищет такую точку входа в модулях личной (если она назначена), а потом системной библиотеки RL и присоединяет нужные модули к фазе (если программистом не введен режим NOAUTO).

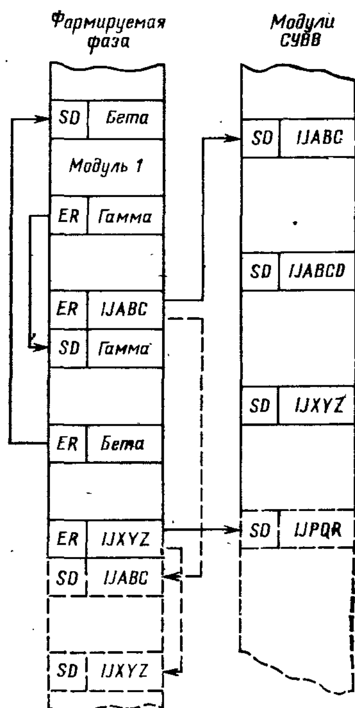


Рис. 6.3. Формирование передач управления при редактировании программы.

Общие области РЕДАКТОР формирует в порядке поступления определяющих их элементов CM и устанавливает их длины как максимальные из длин во всех CM, задающих область с одним и тем же именем.

Неименованные программные секции, определяемые

элементами PC, РЕДАКТОР размещает в программе в порядке их появления в просматриваемых модулях. Адрес начала такой секции нужен только внутри ее текста, так как извне управление этой секции передано быть не может.

Входное имя, содержащееся в данном модуле и определяемое элементом LD, при редактировании может получить номер элемента SD, CM или PC своего модуля, если определяемые ими имена совпадают. Если же в пределах модуля совпадающих имен РЕДАКТОР не обнаруживает, то он ищет такие имена в других модулях по их элементам ER и присоединяет эти модули к формируемой фазе.

6.4. Операторы РЕДАКТОРА

При создании программы РЕДАКТОР должен располагать информацией:

— о фазах, составляющих программу, их размещении в памяти и порядке следования;

— о модулях, составляющих каждую из фаз;

— о точках входа в фазы, если они не расположены в начале фаз;

— о желательном программисту режиме редактирования.

Всю эту информацию программист задает РЕДАКТОРУ в виде последовательности управляющих операторов РЕДАКТОРА, имеющих формат

<имя оператора> <список операндов>

где, как и в других случаях

<список операндов> : : =<операнд> | <операнд>, <список операндов>

Пробел в тексте операндов означает конец списка, что следует учитывать при написании и кодировании операторов РЕДАКТОРА.

Операторы поступают РЕДАКТОРУ с устройства SYSLNK, куда они передаются с SYSRDR, либо из биб-

лиотек объектных модулей, либо, наконец, с SYSIPT (если они входят в состав смешанных модулей).

РЕДАКТОР ДОС/ЕС пользуется операторами ACTION, PHASE, INCLUDE и ENTRY.

6.4.1. Оператор ACTION. Этот оператор устанавливает режим работы РЕДАКТОРА. Он необязателен, но если один или несколько таких операторов использованы, то они должны предшествовать любому из операторов PHASE и INCLUDE, а также любому из включаемых объектных модулей.

Формат оператора ACTION

ACTION <режим>

где

<режим> : : = CLEAR|MAP|NOMAP|NOAUTO|CANCEL|F1|F2

В режиме CLEAR РЕДАКТОР всю свободную часть библиотеки CL заполняет 16-ричными нулями (X'00'), что значительно замедляет процесс редактирования. Поэтому применять этот режим без особой необходимости не рекомендуется.

Режим MAP обеспечивает в процессе редактирования выдачу на системную печать обзора фаз и диагностических сообщений РЕДАКТОРА. Для его реализации требуется наличие назначения для SYSLST, но при этом в ДОС/ЕС предполагается действие режима MAP и по умолчанию.

Режим NOMAP позволяет производить печать обзора фаз на SYSLST, а печать диагностических сообщений РЕДАКТОРА переносит на SYSLOG. Если же для SYSLST физическое УВВ не назначено, то режим NOMAP действует по умолчанию.

Режим NOAUTO отменяет стандартно действующий в системе режим AUTOLINK, в условиях которого РЕДАКТОР самостоятельно пытается разрешить все внешние связи модулей, в том числе и за счет поиска необходимых для этого модулей из личной и системной библиотек RL, не указанных программистом (обычно это модули системы управления вводом — выводом).

Режим CANCEL обеспечивает прекращение редакти-

рования при обнаружении ошибок, которые в диагностических сообщениях РЕДАКТОРА кодируются номерами 2100I—2179I, и разрешает продолжение редактирования при обнаружении ошибок, закодированных другими номерами.

Режимы F1 и F2 служат для редактирования программ разделов переднего плана.

Используя перечисленные режимы редактирования, программист может не заботиться об адресах размещения программ. Память будет распределена в соответствии с тем распределением разделов, которое действует в момент редактирования.

Оператор ACTION может иметь только один операнд. Поэтому если необходимо, например, ввести режим NOAUTO для редактирования программы раздела F2, то следует записать

```
ACTION NOAUTO
ACTION F2
```

Пример 50. Создать и выполнить однофазную программу для раздела F2 переднего плана из исходного модуля на ФОРТРАНе. Предусмотреть режим отмены задания при наличии ошибок редактирования.

Решение.

```
// JOB ПРИМ50
   ACTION F2
   ACTION CANCEL
   PHASE <операнды оператора PHASE>
* (см. ниже п.6.4.2)
// OPTION CATAL
// EXEC FORTRAN
   <перфокарты исходного модуля>
/*
// EXEC LNKEDT
// EXEC <имя фазы>
   <перфокарты с исходиыми данными>
/*
/&
```

6.4.2. Оператор PHASE. Оператор PHASE раскрывает возможности РЕДАКТОРА ДОС/ЕС в части построения многофазных программ. Его формат

PHASE<имя фазы>,<адрес загрузки>[,NOAUTO]

и в последовательности загружаемых операторов его

следует помещать перед первым из включаемых в программу объектных модулей. Если это требование нарушено и объектный модуль попал на SYSLNK до поступления оператора PHASE, то РЕДАКТОР строит безымянную фазу и не каталогизирует ее, даже если ранее был затребован режим CATAL. И наоборот, если после оператора PHASE и до появления следующего такого оператора или до оператора

// EXEC LNKEDT

на SYSLNK не было принято ни одного объектного модуля, то РЕДАКТОР строит пустую фазу.

Каждый оператор PHASE может объединять в одну фазу и несколько модулей, как указанных программистом явно, так и не указанных им, но необходимых для разрешения внешних связей объединяемых модулей.

Первый операнд оператора PHASE

<имя фазы>::=<набор от одного до восьми символов, первый из которых не может быть звездочкой>

Применение букв русского алфавита в имени фазы допускается.

Для программ, состоящих из нескольких фаз, для удобства работы СУПЕРВИЗОРА ДОС/ЕС желательно, чтобы первые четыре символа имен фаз, входящих в такие программы, совпадали. Если это требование не выполнено, то построенная программа работать будет, но процесс вызова некоторых ее фаз займет больше времени.

Для программ, предназначенных для работы в разделах переднего плана, на имена фаз налагается дополнительные ограничения — первые три символа их имен должны быть FGP.

Второй операнд — «адрес загрузки» позволяет строить разнообразные по структуре многофазные программы: с перекрытием, с корневой фазой и т. п.

Имеется шесть различных форм написания этого операнда:

<адрес загрузки>::=S[+<смещение>]|<символическое имя>[(<имя фазы>)][±<смещение>] |ROOT|*[±<смещение>]|+<смещение>| F+<смещение>

Рассмотрим эти формы.

Первая форма представления адреса загрузки:

PHASE FGPD1, S

т. е. фаза FGPD1 при загрузке должна быть размещена, начиная с адреса S, причем значение S зависит от наличия в операторе ACTION операндов F1 или F2. При наличии одного из этих операндов S представляет собой адрес соответствующего раздела переднего плана, увеличенный на сумму длин области сохранения, области меток и общей области. Если же в операторе ACTION операндов F1 или F2 нет, то S означает адрес первого двойного слова после конца области оперативной памяти, занимаемой СУПЕРВИЗОРОМ.

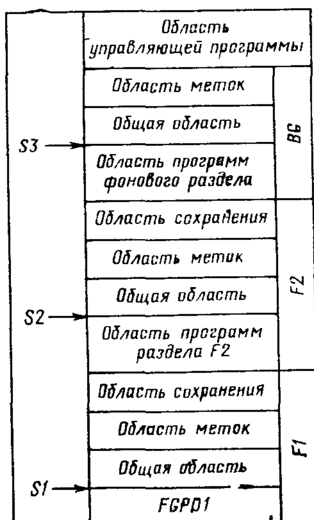


Рис. 6.4. Структура памяти при многопрограммном режиме работы.

Например, при
ACTION F1
PHASE FGPD1, S

адрес загрузки фазы FGPD1 (рис. 6.4) S=S1. При
ACTION F2

S=S2 и, наконец, при отсутствии в операторе ACTION операндов F1 или F2 или отсутствии оператора ACTION вообще (так как наличие его при редактировании необязательно) S=S3.

Пример 51. Создать и каталогизировать под именем FGP12 однофазную программу для работы в разделе F2 переднего плана из исходного модуля на языке ФОРТРАН-4.

Решение.

```
// JOB ПРИМ51
// OPTION CATAL
   ACTION F2
   PHASE FGP12,S
// EXEC FFOPTRAN
   <п/к исходного модуля>
/*
// EXEC LNKEDT
/&
```

Полученная программа при ее выполнении будет вызываться по адресу S2 (рис. 6.4).

Запись

PHASE ABCDE, S+<смещение>

позволит между общей областью и началом размещаемой фазы оставить свободный участок памяти (рис. 6.5), причем

<смещение>::=<16-ричное число>|<десятичное число>|nK

16-ричное число пишется, как обычно, в виде

X'hhh....h'

с количеством цифр не более шести и

h::=<десятичная цифра>|A|B|C|D|E|F

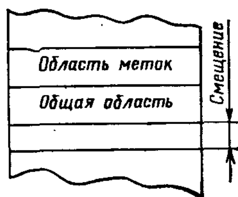


Рис. 6.5. Фаза, загружаемая со смещением.

Изображение десятичного числа должно содержать не более восьми десятичных цифр. Наконец, nK означает, что смещение производится на целое число n областей по K=1024 слова в каждой.

Пример 52. Фазы FGPD1 и FGPD2 должны попеременно работать в разделе F1 переднего плана, пользуясь материалами, поставляемыми им фазой FGPD3. После окончания их работы вызывается фаза FGPD4, которая работает самостоятельно. Построить адреса загрузки этих фаз, если F1 имеет объем 10K, конец общей области лежит на расстоянии $l \leqslant 1K$ слов от начала раздела, а строящиеся фазы будут занимать не свыше 3K, 4K, 5K и 8K слов.

Решение. Память раздела больше, чем 9K слов. Поэтому фазы FGPD1 и FGPD2, работающие попеременно, можно разместить с адреса S, а фазу FGPD3 — со смещением на 4K. При этом следует применить операторы УЗ и РЕДАКТОРА в таком порядке.

```
// JOB ПРИМ52
// OPTION CATAL
   ACTION F1
   PHASE FGPD1,S
// EXEC ASSEMBLY
   <перфокарты FGPD1>
/*
   PHASE FGPD2,S
// EXEC ASSEMBLY
   <перфокарты FGPD2>
/*
```

```
   PHASE FGPD3,S+4K
// EXEC ASSEMBLY
   <перфокарты FGPD3>
/*
   PHASE FGPD4,S
// EXEC ASSEMBLY
   <перфокарты FGPD4>
/*
// EXEC LNKEDT
/&
```

Размещение фаз в памяти для раздела F1 на разных этапах работы фаз показано на рис. 6.6.

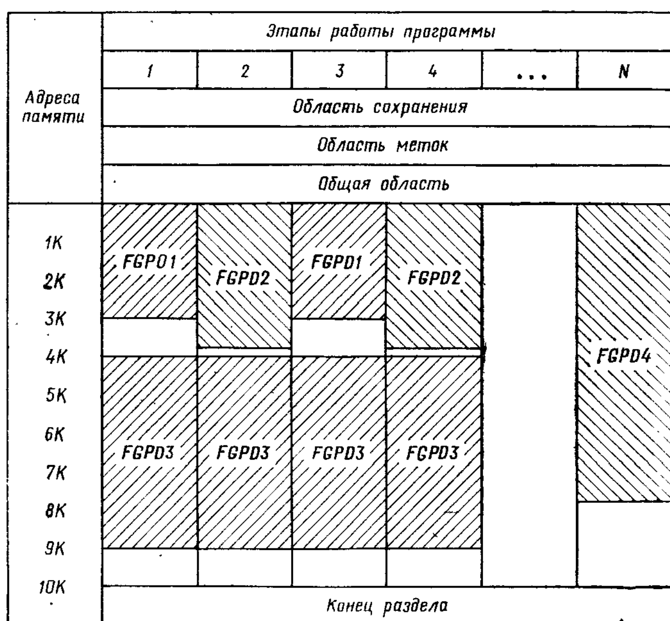


Рис. 6.6. Размещение фаз в примере 52.

Вторая форма представления адреса загрузки:

PHASE ABCD2, ABCD1

При этом фаза ABCD2 загружается так, что ее начало совпадает с началом размещения фазы ABCD1. Так, если для ABCD1 был указан адрес S, то по этому же адресу разместится и ABCD2.

При

PHASE ABCD2,XYZZZ(ABCD1)

фаза ABCD2 размещается, начиная с того адреса, который во время размещения ABCD1 занимала ее секция или подпрограмма XYZZZ. Перекрыта, таким образом, будет не вся фаза ABCD1, а только ее часть (см. рис. 6.7).

Формы написания

PHASE ABCD2,ABCD1±смещение

и

PHASE ABCD2,XYZZZ(ABCD1)±<смещение>

позволяют сдвинуть загружаемую фазу как в сторону больших, так и в сторону меньших адресов, что было невозможно при загрузке по адресу S, так как приводило бы к порче текста СУПЕРВИЗОРА и другой информации.

В общем случае написание

PHASE<имя фазы>,<символическое имя>(<имя фазы>)±<смещение>

позволяет в качестве символического имени использовать не только имена программных секций.

Имеется возможность размещать очередную фазу как с первого адреса другой указанной фазы, так и с адресов ее именованных составных частей, или сдвигать этот начальный адрес на величину смещения.

До перекрытия	После перекрытия
Фаза ABCD1, программная секция XY	Фаза ABCD1, программная секция XY
Программная секция XYZ	Программная секция XYZ
Программная секция XYZZ	Программная секция XYZZ
Программная секция XYZZZ	Фаза ABCD2
Программная секция XYZZZZ	

Рис. 6.7. Перекрытие отдельных программных секций фазы.

Пример 53. Построить программу, в которой первая фаза AABB11 состоит из секций AX11,AX12,AX13, а вторая фаза AABB22 на секции не делится, причем вторая фаза должна вызываться на место секции AX13.

Решение.

```
// JOB ПРИМ53
// OPTION CATAL
```

```
   PHASE AABB11,S
// EXEC ASSEMBLY
   <перфокарты AABB11>
* ЗДЕСЬ ТРАНСЛИРУЕТСЯ ИСХОДНЫЙ МОДУЛЬ НА ЯЗЫКЕ
* АССЕМБЛЕРА, ОБРАЗУЮЩИЙ ПРИ РЕДАКТИРОВАНИИ
* ФАЗУ AABB11
/*
   PHASE AABB22,AX13(AABB11)
// EXEC ASSEMBLY
   <перфокарты AABB22>
/*
// EXEC LNKEDT
/&
```

Третья форма представления адреса загрузки:

PHASE ABCDE, ROOT

употребляется только для редактирования фазы ABCDE как корневой, и всякое размещение другой фазы на ее месте считается ошибкой (корневой называется фаза, вызываемая первой и остающаяся в памяти на протяжении всего времени работы программы). Операнд ROOT может присутствовать только в первом из операторов PHASE. Появление его в любом другом месте рассматривается не как адрес, а как имя фазы.

Адрес загрузки корневой фазы в памяти раздела совпадает с адресом, определяемым по операнду S.

Пример 54. Фаза AB121 должна работать попеременно с фазами AB122, AB123 и AB124, каждая из которых наличия в памяти двух других не требует. Необходимо написать операторы РЕДАКТОРА для каталогизации этой четырехфазной программы, считая, что фаза AB121 не займет более 4K слов от начала раздела.

Решение.

```
// JOB ПРИМ54
// OPTION CATAL
   PHASE AB121,ROOT
// EXEC PL/I
   <перфокарты AB121>
/*
   PHASE AB122,S+4K
// EXEC PL/I
   <перфокарты AB122>
/*
   PHASE AB123,AB122
// EXEC ASSEMBLY
   <перфокарты AB123>
/*
   PHASE AB124,AB123
// EXEC ASSEMBLY
```

```
    <перфокарты AB124>
/*
// EXEC LNKEDT
/&
```

Четвертая форма представления адреса загрузки:

PHASE<имя фазы>, * [±<смещение>]

При записи

PHASE ABCDE, *

фаза ABCDE размещается сразу за загруженной фазой, начиная с первого свободного двойного слова. Если фаза ABCDE размещается первой, то операнды * и S обеспечивают одинаковое ее размещение.

Наличие во втором операнде смещения, например,

PHASE XYZ22, * +X'12FF'

приводит к таким же сдвигам в размещении фазы, как и при использовании второй формы представления.

Пример 55. Построить такую программу, чтобы фаза 1 была корневой, фазы 2 и 3 работали поочередно, а во время работы фазы 3 могли бы поочередно работать фазы 4 и 5.

Решение. Обозначив фазы 1—5 через ABCD1—ABCD5, построим следующую последовательиость управляющих операторов:

```
// JOB ПРИМ55
// OPTION CATAL
    PHASE ABCD1,ROOT
// EXEC ASSEMBLY
    <перфокарты фазы 1>
/*
    PHASE ABCD2,*
// EXEC ASSEMBLY
    <перфокарты фазы 2>
/*
    PHASE ABCD3,ABCD2
// EXEC ASSEMBLY
    <перфокарты фазы 3>
/*
    PHASE ABCD4,*
// EXEC ASSEMBLY
    <перфокарты фазы 4>
/*
    PHASE ABCD5,ABCD4
// EXEC ASSEMBLY
    <перфокарты фазы 5>
/*
// EXEC LNKEDT
/&
```

Расположение фаз в памяти машины при таком редактировании показано на рис. 6.8. Звездочка в случае фазы 2 ставит ее вслед за корневой фазой, а в случае фазы 4 — за находящимися в памяти фазами 1 и 3.

Пятая форма представления адреса загрузки:

PHASE<имя фазы>, +<смещение>

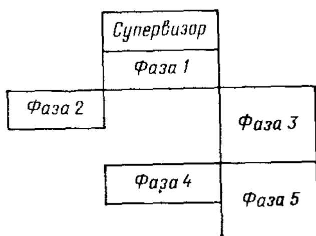


Рис. 6.8. Программа оверлейной структуры.

В этом случае программист задает абсолютный (отсчитанный от начала оперативной памяти) адрес размещения фазы. Например, зная, что длина выбранного раздела памяти равна 20К байтов и что остальные фазы, одновременно присутствующие в памяти, занимают не свыше 10К байтов, программист может для фазы AAABBB написать

.PHASE AAABBB,+12K

Шестая форма представления адреса загрузки:

PHASE<имя фазы>,F+<смещение>

Употребляется эта форма, когда редактирование фазы для переднего плана идет при наличии в системе только фонового раздела, когда мультипрограммный режим не используется или когда предполагается использовать фазу при отличном от действующего распределения памяти между разделами. Величина F — символ, указывающий РЕДАКТОРУ, что фаза редактируется для раздела, адрес которого задан <смещением>. РЕДАКТОР настраивает фазу на адрес

<смещение>+<сумма длин областей сохранения и меток>+<длина общей области>

Пример 56. Отредактировать две фазы, которые предполагается использовать на другой вычислительной установке: одну с адреса 18К, а другую вслед за ней.

Решение.

```
// JOB ПРИМ56
   PHASE AABBCC,F+18K
// OPTION CATAL
// EXEC FORTRAN
   <перфокарты исходного модуля>
/*
```

```
   PHASE AABB22,*
// EXEC FORTRAN
   <перфокарты исходного модуля>
/*
// EXEC LNKEDT
// EXEC CSEPV
* ЭТОЙ ПРОГРАММОЙ БИБЛИОТЕКАРЯ ФАЗЫ БУДУТ ОТ-
* ПЕРФОРИРОВАНЫ.
   PUNCH AABB.ALL
/*
/&
```

Как уже было сказано в п. 5.5.2, каждая из фаз выдается в абсолютных адресах, т. е. в 16-ричном формате, перед которым будет сформирована карта ESD, а вслед за этим карта END (это позволит нам в следующем разделе построить пример включения этих фаз в состав библиотеки абсолютных модулей той установки, для которой они готовились).

Последним и необязательным операндом оператора PHASE является операнд NOAUTO, отделяемый от предыдущего запятой. Действие этого операнда аналогично действию такого же операнда в операторе ACTION и состоит в отмене режима AUTOLINK, но здесь эта отмена распространяется только на время редактирования одной фазы, а не всей программы.

Пример 57. Построить фазу из объектного модуля, поставляемого транслятором ПЛ/1, и при редактировании запретить присоединение системных модулей (модулей ввода — вывода, элементарных функций, преобразования данных из одной формы представления в другую и пр.)

Решение.

```
// JOB ПРИМ57
// OPTION LINK
   PHASE XXCCXX,S,NOAUTO
// EXEC PL/I
   <перфокарты исходного модуля>
/*
// EXEC LNKEDT
/&
```

6.4.3. Оператор INCLUDE. В то время как операторы PHASE определяют фазовую структуру программы, операторы INCLUDE определяют модульную структуру фаз, задают для каждой фазы набор модулей или входящих в них программных секций.

Формат оператора

INCLUDE<имя модуля>, (<список секций>)

Все модули или их программные секции включаются в фазу, определенную оператором PHASE, стоящим перед операторами INCLUDE, вызывающими модули. Для включения в фазу модуля с SYSIPT используется форма оператора INCLUDE без операндов, и в этом случае в отличие от операторов с операндами он обрабатывается не РЕДАКТОРОМ, а программой УПРАВЛЕНИЕ ЗАДАНИЯМИ.

В отличие от других операторов РЕДАКТОРА оператор INCLUDE на SYSLNK не переносится, а только обеспечивает перенос туда соответствующего объектного модуля или некоторых его секций.

Если отсутствует второй операнд, то модуль включается в фазу целиком. В противном случае включаются только указанные во втором операнде секции (не больше пяти).

При необходимости включить большее число секций используются два и более оператора INCLUDE с тем же именем модуля. Если включению в фазу подлежат секции модуля, расположенного на SYSIPT, то в операторе присутствует только второй операнд

INCLUDE,(список секций)

перед которым ставится запятая.

На SYSLNK в этом случае модуль переносится целиком и лишь при редактировании из него исключаются ненужные секции. Поиск модуля, указываемого в первом операнде INCLUDE, производится сначала в личной библиотеке RL, если для нее есть назначение, а в случае его отсутствия в личной библиотеке поиск продолжается в системной библиотеке.

Оператор INCLUDE может считываться РЕДАКТОРОМ с SYSRDR и SYSLNK, а также из системной и личных библиотек RL (из имеющихся в этих библиотеках нестандартных модулей). В первых двух случаях вызываемые модули именуются модулями первого уровня, в последнем случае это модули второго, третьего и более высоких уровней. Всего РЕДАКТОР ДОС/ЕС может работать с модулями не более шести уровней.

Пример 58. Создать однофазную программу, включнв в нее из системной библиотеки RL модули XYZ1 и XYZ2, модуль, написанный на ФОРТРАНе, а также 2-ю и 3-ю секции модуля, находящегося на SYSIPT. Созданной программе передать управление.

Решение.

```
// JOB ПРИМ58
// OPTION LINK
   INCLUDE XYZ1
   INCLUDE XYZ2
// EXEC FORTRAN
   <исходный модуль>
/*
   INCLUDE,(SECT2, SECT3)
   <перфокарты объектного модуля>
/*
// EXEC LNKEDT
// EXEC
   <Исходные данные>
/*
/&
```

Пример 59. Создать трехфазную программу для работы в первом разделе переднего плана, включив в корневую фазу два модуля с SYSIPT, во вторую фазу модули AB12, AB13 и AB14, а в третью — модули AB15 и AB16.

Решенне.

```
// JOB ПРИМ59
// OPTION CATAL
   ACTION F1
   PHASE FGP11,ROOT
   INCLUDE
   <первый модуль>
/*
   INCLUDE
   <второй модуль>
/*
   PHASE FGP12,*
   INCLUDE AB12
   INCLUDE AB13
   INCLUDE AB14
   PHASE FGP13,FGP12
   INCLUDE AB15
   INCLUDE AB16
// EXEC LNKEDT
/&
```

Пример 60. Создать двухфазную программу с перекрытием. Первую фазу составить из модулей F.ABC1, ABC2, ABC3 и ABC4. Модуль ABC1 на языке ФОРТРАН взять из системной библиотеки SL, протранслировать и исходный модуль удалить. Модуль ABC2 взять из системной библиотеки RL, потом его удалить, а RL сжать. ABC3 взять с SYSIPT и транслировать его с ФОРТРАНа, а ABC4 взять также с SYSIPT в готовом виде. Вторую фазу составить из

модулей XYZ1 и XYZ2, первый из которых находится на SYSIPT в готовом виде, а второй ждет трансляции. Оба модуля необходимо также каталогизировать в библиотеке RL.

Решение.

```
// JOB ПРИМ60
// OPTION CATAL
// ASSGN SYSPCH.X'281'
// MTC REW,X'281'
// MTC WTM.SYSPCH
// EXEC JSERV
   PUNCH F.ABC1
/*
// MTC WTM, SYSPCH
// MTC REW,X'281'
// ASSGN SYSPCH, UA
   PHASE MPX11,
// ASSGN SYSIPT.X'281'
// EXEC FORTRAN
// EXEC MAINT
   DELETS F.ABC1
/*
   INCLUDE ABC2
// EXEC MAINT
   DELETR ABC2
   CONDS RL
/*
// EXEC FORTRAN
   <перфокарты ABC3>
/*
   INCLUDE
   <перфокарты ABC4>
/*
   PHASE MPX12,MPX11
   INCLUDE
   <перфокарты модуля XYZ1>
/*
// EXEC ASSEMBLY
   <перфокарты модуля XYZ2>
/*
// EXEC MAINT
   CATALR XYZ1, SYSLNK
   CATALR XYZ2, SYSLNK
// EXEC LNKEDT
/*
/&
```

6.4.4. Оператор ENTRY. Этот оператор не является обязательным в цепочке операторов РЕДАКТОРА, но если он присутствует то это означает, что информация РЕДАКТОРА исчерпана. Дополнительно он может определить точку входа в создаваемую программу. Его формат —

ENTRY<имя точки входа>

Оператор ENTRY считается с SYSRDR программой УПРАВЛЕНИЕ ЗАДАНИЯМИ и переносится на SYSLNK, где его и обнаруживает РЕДАКТОР. Если на SYSRDR программа УПРАВЛЕНИЕ ЗАДАНИЯМИ его не обнаружит, то на SYSLNK она этот оператор построит самостоятельно, так что для РЕДАКТОРА он всегда существует. Таким образом, оператор ENTRY необходим только для задания точки входа в первую фазу программы, если она не совпадает с ее началом.

7. Управление данными

7.1. Система управления вводом—выводом

Система управления вводом — выводом (СУВВ) ДОС/ЕС — это комплекс самоперемещающихся подпрограмм — транзитов, которые можно использовать в любом из разделов памяти. Хранятся эти подпрограммы в библиотеке CL и по своему назначению делятся на две группы.

Подпрограммы первой группы предназначены для управления конкретными физическими УВВ; они учитывают их свойства и возможности и обеспечивают запуск и выключение. Эти подпрограммы составляют *физическую систему управления вводом — выводом* (ФСУВВ), являющуюся частью СУПЕРВИЗОРА.

Подпрограммы второй группы связывают конкретную программу с ФСУВВ, когда возникает необходимость обмена данными между оперативной памятью и внешними носителями. Эти подпрограммы составляют *логическую систему управления вводом — выводом* (ЛСУВВ). ФСУВВ в конечном счете используется всеми программами, нуждающимися в пересылке данных за пределы или из-за пределов оперативной памяти. Для ФСУВВ безразлично логическое содержание данных, их формат и организация. Ее функция — следить за выполнением программы канала, которую формирует либо программист, либо ЛСУВВ.

ФСУВВ работает совместно с *планировщиком каналов*, вход в который (см. § 1.7) осуществляется по команде SVC (вызов СУПЕРВИЗОРА).

Для программиста-пользователя обращение к ФСУВВ является оправданным только тогда, когда ресурсы алгоритмических языков или макрокоманд АССЕМБЛЕРА не позволяют использовать возможностей ЭВМ, без которых задача не решается. Для использования ФСУВВ программист должен знать не только особенности ДОС/ЕС, но и принципы управления конкретными устройствами ввода — вывода.

ЛСУВВ состоит из подпрограмм, которые, с одной стороны, связаны с программой пользователя, а с другой — с ФСУВВ.

Основные функции ЛСУВВ:

— запрос ФСУВВ и определение ее задач;

— обработка состояний «конец файла» и «конец тома»;

— соединение записей в блоки при выводе и разделение блоков на записи при вводе;

— управление областями ввода — вывода (построение буферов).

ЛСУВВ рассматривает файлы как логические структуры данных. Ее модули могут включаться в программу пользователя двумя способами:

— в процессе трансляции вместе с текстом исходной программы, написанной на языке АССЕМБЛЕРА;

— в процессе редактирования путем присоединения к программе готовых модулей из библиотеки RL, что достигается включением в программу пользователя операторов ввода — вывода, описанных в гл. 4.

Наибольшие возможности применения системы команд машины дает первый способ включения модулей ЛСУВВ, но его недостатком является необходимость затрат машинного времени на их трансляцию и невозможность использования иначе как при написании программ на языке АССЕМБЛЕРА.

Использование готовых модулей ЛСУВВ из библиотеки RL, экономя время программиста, ограничивает его возможности только использованием устройств, для которых модули ЛСУВВ имеются в библиотеке RL. Для работы с ФСУВВ программист должен располагать сведениями о характеристиках УВВ и размещении на них данных.

Для работы с ЛСУВВ необходимо знать имя файла, тип УВВ, способ организации данных, метод доступа, формат и длину записей.

При программировании на языке АССЕМБЛЕРА эта информация дается в качестве макрокоманд DTF. Для языков ФОРТРАН и ПЛ/1 операторы, необходимые для управления данными, изложены в гл. 4.

7.2. Организация данных на внешних носителях ЕС ЭВМ

Описанная в § 1.2 организация данных в ЕС ЭВМ предполагает их разбиение на порции различной величины, называемые *записями, блоками, файлами* и *томами*. Такая организация принята для всех видов УВВ, используемых ЭВМ Единой Системы.

Для различных носителей характерны некоторые ограничения организации наносимой на них информации, хотя это в большей степени относится не к носителям, а к действующей на ЭВМ системе управления вводом — выводом.

7.2.1. Организация данных иа магнитных лентах. Бобина магнитной ленты считается в ДОС/ЕС томом. Для опознания тома вне зависимости от того, имеются на бобине какие-либо «внешние» метки — знаки, сделанные карандашом, краской или еще чем-нибудь, или таких меток нет, в ДОС/ЕС принято снабжать тома магнитных лент «внутренними» метками— специальными записями, наносимыми магнитным способом на определенные места ленты.

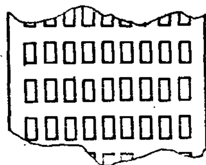


Рис. 7.1. Схема записи данных на магнитной ленте.

Как известно, на магнитных лентах информация располагается в виде дорожек. Для ЕС ЭВМ используются девятидорожечные магнитные ленты, причем на каждой из этих дорожек располагаются площадки, которые могут намагничиваться. Различаются два состояния намагничивания этих площадок, которые истолковываются как двоичные цифры 0 и 1. На девяти площадках, принадлежащих различным дорожкам, размещенным вдоль одного перпендикуляра к краю ленты (рис. 7.1), помещается один байт информации и контрольный разряд. Плотность размещения информации на магнитной ленте ЕС ЭВМ составляет 32 или 64 байта на миллиметр.

Записи на магнитной ленте отделяются друг от друга промежутками, доля которых при использовании коротких записей в общей длине ленты может быть достаточно большой. Поэтому принято короткие записи объединять в группы, называемые блоками. Иногда записи называют *логическими записями*, а блоки — *физическими записями*.

Файлы на магнитных лентах представляют собой объединение подряд расположенных записей или блоков. Для облегчения использования файлы могут снабжаться опознавательными записями — *метками*, а также специальными разделительными записями — *маркерами* или ленточными марками. Метки и маркеры могут располагаться перед и после файла, но допустимо в ДОС/ЕС использование файлов, не имеющих не только меток, но и маркеров.

7.2.2. Организация даиных иа магнитных дисках. На магнитных дисках информация располагается байт за байтом вдоль концентрических дорожек; продолжение записи, не поместившейся на дорожке, заносится не на соседнюю дорожку того же диска, а на дорожку соседнего диска, имеющую тот же номер, считая от края диска. В дисковых пакетах ЕС ЭВМ (рис. 7.2) десять рабочих поверхностей дисков, на каждой из которых по 203 дорожки (дорожки 0—199 основные, 200—202 запасные). Каждые десять дорожек образуют *цилиндр*. Цилиндры, как и дорожки, нумеруются от края пакета к центру. Дорожки же в пакете имеют сквозную нумерацию типа $N=10C+P$, где C — номер цилиндра ($0 \leqslant C \leqslant 202$), а P — номер рабочей поверхности ($0 \leqslant P \leqslant 9$).

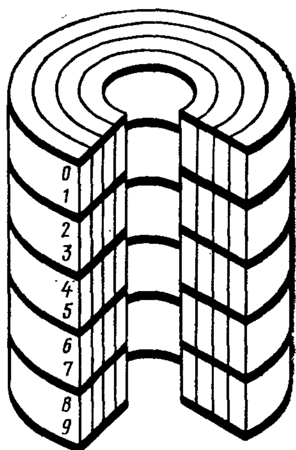


Рис. 7.2. Структура пакета дисков.

Начинается каждая дорожка маркером, наносимым аппаратным способом.

Метки на дисках не могут физически соседствовать с файлами, так как последние могут размещаться не

только на соседних дорожках, но и другими способами (см., например, ниже «индексно-последовательные файлы»). Поэтому в ДОС/ЕС метки файлов на дисках принято помещать вместе с меткой тома (пакета дисков) в виде специального файла, называемого *оглавлением тома*, в определенном месте этого тома.

7.2.3. Типы записей в ДОС/ЕС. В ДОС/ЕС имеется три типа записей: записи *фиксированной* длины, записи *переменной* длины и записи *неопределенной* длины.

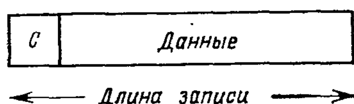


Рис. 7.3. Запись фиксированной длины.

Запись фиксированной длины кроме области данных может содержать управляющий символ, занимающий 1 байт (на рис. 7.3 обозначен буквой С), позволяющий изменять режимы печати и перфорации данных. Длина записи, включая символ С, если он есть, в самой записи не указывается и поступает к ЛСУВВ от программы пользователя.

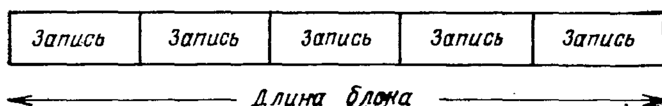


Рис. 7.4. Блок записей фиксированной длины.

При соединении записей фиксированной длины в блок необходимо, чтобы количество записей в каждом блоке данного файла было одинаковым. Исключение составляет последний блок, для которого это ограничение не обязательно. Длина блока сообщается системе теми же средствами, что и длина записи. В самом блоке места для этой информации не отводится.

Вид блока записей фиксированной длины показан на рис. 7.4.

Записи и блоки фиксированной длины наиболее удобны для манипулирования с ними, но их использование может иногда привести к нерациональному расходованию памяти. Во избежание этого применяют записи переменной длины, строение которых показано на рис.

7.5. Запись в этом случае кроме данных и байта с символом С занимает четыре байта, первые два из которых содержат длину записи в байтах (включая и эти байты),



Рис. 7.5. Запись переменной длины.

а вторые два заполняются нулями. Пользуясь первыми байтами, система распознает длину записи (до 65535 байтов), получая от программы пользователя только информацию о типе записи.

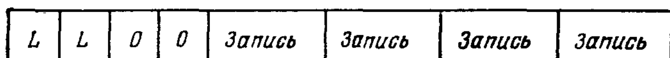


Рис. 7.6. Блок записей переменной длины.

Для записей переменной длины в отличие от записей фиксированной длины блокирование допустимо и при разной длине записей. Структура блока записей переменной длины показана на рис. 7.6.

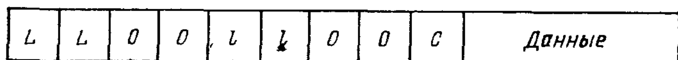


Рис. 7.7. Блок в составе одной записи переменной длины.

В таком блоке байты LL, как и в записи, содержат полную длину блока в байтах, а последующие два — нули. Неблокированная запись считается включенной в блок с одной записью и поэтому имеет вид, показанный на рис. 7.7.



Рис. 7.8. Запись неопределенной длины.

ДОС/ЕС позволяет оперировать и с записями, структура и длина которых неизвестна. Для этих целей вводится понятие записи неопределенной длины (рис. 7.8). Записи этого вида в бло-

ки не объединяются, а длину их система определяет в процессе ввода — вывода, обнаруживая конец информации.

Возможность использования различных типов записей на УВВ ЕС ЭВМ и налагаемые этими устройствами ограничения показаны в табл. 7.1.

Таблица 7.1

Тип устройств ввода—вывода	Допустимая длина записи в байтах	Допускаются ли записи				
		фиксированной длины		переменной длины		неопределенной длины
		неблокированные	блокированные	неблокированные	блокированные	неблокированные
Пишущая машинка	256	Да	Нет	Нет	Нет	Да
АЦПУ	Не более длины строки	Да	Нет	Да	Нет	Да
Ввод с перфокарт	80	Да	Нет	Нет	Нет	Нет
Вывод с перфокарты	Не ограничена	Да	Нет	Да	Нет	Да
Магнитные ленты и диски						
уровень GET-PUT	»	Да	Да	Да	Да	Да
уровень READ-WRITE	»	Да	Да	Да	Да	Да

7.3. Методы логической организации данных

Переходя к описанию конкретных методов логической организации данных — организации файлов, укажем на соотношение между методами организации файлов и методами доступа к ним.

В ДОС/ЕС применяются три метода *организации* файлов: *последовательный, прямой* (в ПЛ/I он именуется *региональным*) и *индексно-последовательный* (при доработках системы ожидается расширение этого списка), тогда как методов *доступа* только два: *последовательный* и *прямой*. (Заметим, что в ОС/ЕС ЭВМ классификация методов доступа совершенно иная.)

7.3.1. Последовательная организация файлов. При последовательной организации файлов предполагается, что записи вносятся в файл и считываются из него в строго последовательном порядке: (N+1)-я запись может быть занесена или прочитана только после просмотра всех предыдущих N записей. Возвратов во время работы с файлом не допускается. Наряду с прямой обработкой для файлов на магнитной ленте допустима обработка в обратном направлении, но в этом случае уже без каких-либо изменений этого направления на прямое.

Такой метод организации файлов оправдывает себя в случаях, когда все или большая часть записей должна обрабатываться в том порядке, в каком они располагаются в файле. Следует подчеркнуть, что последовательная организация файлов предполагает и последовательный метод доступа к ним.

Используется такая организация:

— при вводе данных с перфокарт и выводе на перфокарты;

— при работе с магнитными лентами;

— при выдаче данных на АЦПУ.

Можно применять эту организацию и для файлов на дисках.

Хотя при работе с файлами на магнитных лентах нет возможности внутри программы делать в файле возвраты, пропуски и т. д., но при переходе от одного шага задания к другому такая возможность реализуется оператором УПРАВЛЕНИЕ ЗАДАНИЯМИ

// MTC <код команды>,<УВВ> [,nn]

В этом операторе

<код команды>: :=BSF|BSR|ERG|FSF|FSR|RUN|REW|WTM

<УВВ>: :=X'cuu'|SYSxxx

Здесь nn — не более чем двухзначное десятичное число.

Различные коды команды позволяют выполнить следующие действия:

BSF — шаг на группу блоков назад до маркера, т. е. до начала файла, до начала области меток и т. п.;

BSR — шаг на один блок назад;

ERG — стереть блок;

FSF — шаг на группу блоков вперед до маркера;

FSR — шаг на один блок вперед;
RUN — перемотать и разгрузить;
REW — перемотать;
WTM — записать маркер.

Число nn, если оно присутствует среди операндов, означает, что команда выполняется несколько раз. При отсутствии nn предполагается, что nn=1.

Пример 61. На устройстве SYS010 определить файл A2112, который должен быть пятым по счету в томе 010101 и храниться один год. Первые четыре файла с маркерами, но без меток. Свободное УВВ имеет адрес X'283'.

Решение. Используя оператор TLBL (см. гл. 2), напишем

```
// JOB ПРИМ61
// ASSGN SYS010,X'283'
// TLBL A2112,365,010101,1,5
// MTC REW,SYS010
// MTC FSF,SYS010,05
. . . . . . . . . . . . .
/&
```

Пример 62. Начать обработку с десятого блока пятого файла, расположенного на SYS010. Между файлами по три маркера (между файлом и метками и между хвостовыми и головными метками разных файлов). Положение ленты неизвестно.

Решение.

```
// MTC REW,SYS010
// MTC FSF,SYS010,14
* ПРОПУСКАЕМ ПЕРВЫЙ, ВТОРОЙ,... ЧЕТВЕРТЫЙ ФАЙЛЫ
* С ИХ МЕТКАМИ И МЕТКИ ПЯТОГО ФАЙЛА
// MTC FSR,SYS010,09
```

7.3.2. Прямая организация файлов. Файлы с прямой организацией могут создаваться только на запоминающих устройствах прямого доступа (в ДОС/ЕС только на дисках). Записи на дисках состоят из области счетчика, области ключа (которая может отсутствовать) и области данных (рис. 7.9). На одной и той же дорожке диска записи могут иметь разную длину как в области ключа, так и в области данных. *Маркер дорожки* указывает ее начало и собственный адрес.

Счетчик	Ключ	Данные

Рис. 7.9. Структура записи на диске.

Собственный адрес дорожки состоит из семи байтов, содержимое которых показано на рис. 7.10. Байт флажка характеризует состояние дорожки. Каждая дорожка начинается со служебной записи R_0, на которой располагаются рабочие записи.

Область счетчика рабочей записи показана на рис. 7.11. Содержимое байта флажка области счетчика записи переносится из байта флажка маркера. В байт «длина ключа» для записей без ключа заносится нуль.

1	2	3	4	5	6	7
Флажок	Номер цилиндра и номер дорожки				Контрольные байты	

Рис. 7.10. Собственный адрес дорожки.

Номер записи в поле идентификатора — это ее физический номер на дорожке. Ключ же является ее логическим номером. Работая на алгоритмических языках, программист может знать только ключ записи и не знать ее номера на дорожке. Область ключа, когда она присутствует, сверх своей длины имеет два контрольных байта, но при указании длины ключа они не учитываются.

1	2	3	4	5	6	7	8	9	10	11
Флажок	Поле идентификатора					Длина ключа	Длина записи		Контрольные байты	
	Номер цилиндра		Номер дорожки		Номер записи					

Рис. 7.11. Область счетчика.

В область данных записывается содержимое блока, т. е. одна или несколько логических записей.

Для подготовки пакета дисков под файл прямой организации требуется его предварительная очистка и разметка с помощью программы CLRDSK, имеющейся в составе библиотеки CL. При разметке создаются области с полем ключа не свыше 256 байтов и полем записи не свыше 65536 байтов. Эти области заполняются символами, которые указывает программист. Размечаемый файл для программы CLRDSK должен именоваться UOUT.

Пример 63. Разметить на пакете № 223112 цилиндры 175 и 176 на области по 400 байтов без поля ключа. Пакет установить на X'192' и присвоить ему имя SYS014. Размечаемые области заполнить двоичными единицами и проверить заполнение считыванием.

Решение.

```
// JOB ПРИМ63
// ASSGN SYS014,X'192'
// DLBL UOUT,74/001
* ДАТА УКАЗЫВАЕТСЯ ПРОШЕДШАЯ, ЧТОБЫ ДОС/ЕС НЕ
* МЕШАЛА ЗАПОЛНЯТЬ ФАЙЛ.
// EXTENT SYS014,223112,1,0,1750,20
// EXEC CLRDSK
// UCL B=(K=0. D=400),X'FF',OY
```

Файлы с прямой организацией делятся на файлы без ключа и файлы с ключом (при создании таких файлов в программе на языке ПЛ/1 им присваиваются описатели REGIONAL(1) и REGIONAL(3)). В файле без ключа записи адресуются по их месту в файле: номеру дорожки и номеру записи внутри дорожки. Файлы с ключами состоят из записей, часть которых выделяется под поле ключа. Записи адресуются номером дорожки и длиной ключа (9—255 символов).

Если заполнение записей в файле без ключа происходит не в порядке их расположения, а произвольным образом, то память под файл должна быть выделена сразу в полном объеме и значительная часть ее долго может не использоваться. Для таких ситуаций более целесообразна прямая организация с ключами или индексно-последовательная организация.

7.3.3. Индексно-последовательная организация файлов. Файлы с индексно-последовательной организацией могут содержать только записи фиксированной длины, соединенные или несоединенные в блоки. При таких файлах создаются *таблицы индексов* разной иерархии. В каждом цилиндре выделяются дорожки для данных, составляющие область данных цилиндра, область переполнения цилиндра, одна или более *дорожек переполнения*. Часть нулевой дорожки отводится под так называемый индекс дорожки.

Для вставки новой записи в индексно-последовательный файл значение ее ключа сравнивается с индексом цилиндра, потом с индексами дорожек, т. е. с наибольшими значениями ключей, имеющихся на дорожках записей. На определенную таким образом дорожку и заносится запись. Место ее на дорожке определяется

также по ключу, и, если необходимо, одна из ранее занесенных записей вытесняется на дорожку переполнения.

Опишем структуру индексно-последовательного файла и порядок внесения в него новых записей.

Имеющаяся на нулевой дорожке каждого цилиндра индексно-последовательного файла служебная запись R_0 используется для указания адреса и размеров области переполнения цилиндра — количества дорожек переполнеиия. Ее структура показана на рис. 7.12.

Байт 0	Байт 1	Байт 2	Байт 3	Байт 4	Байт 5	Байт 6	Байт 7
Номер цилиндра		Номер дорожки		Номер области	Счетчик байтов дорожки переполнения		Не используется

Рис. 7.12. Структура служебной записи R_0.

Вслед за R_0 на нулевой дорожке располагаются парные записи индекса дорожки, по две на каждую дорожку, выделенную для данных. Для дорожек переполнения такие записи не выделяются. Каждая из парных записей индекса дорожки состоит из поля ключа и поля данных. Первая из таких парных записей, описывающих какую-либо дорожку, в поле ключа содержит ключ последней записи данных, присутствующей на дорожке (не обязательно той записи, которая оказалась последней при первичном формировании дорожки, а той, которая оказалась последней после вытеснения некоторых записей на дорожку переполнения, если такое вытеснение имело место). В поле данных этой записи помещается адрес первой записи данных на этой дорожке: для дорожек 1, 2,... это будет адрес первой записи, для нулевой дорожки — адрес той записи, которая непосредственно следует за индексом дорожки. Как и в R_0, этот адрес состоит из номера цилиндра, номера дорожки и номера области.

Вторая запись пары в поле ключа содержит ключ той записи, которая была последней на дорожке в момент ее заполнения. При занесении на дорожку новых записей это поле не меняется. Таким образом, сравнение ключа этих двух записей позволяет установить, были ли случаи вытеснения записей с описываемой ими дорожки.

В поле данных второй записи пары первоначально помещается тот же адрес, который записывается в первую запись. Но затем, после одного или нескольких вытеснений, в этом поле оказывается адрес последней записи, вытесненной с этой дорожки.

Номер дорожки					
	0	Индекс дорожки (поля 1—17)		Поле 18 Запись 5	Поле 19 Запись 8
	1	Поле 1 Запись 26	Поле 2 Запись 31	Поле 3 Запись 40	Поле 4 Запись 45
	2	Поле 1 Запись 64	Поле 2	Поле 3	Поле 4
	3	Поле 1	Поле 2	Поле 3	Поле 4

Рис. 7.13. Пример заполнения индексно-последовательного файла.

Оканчивается индекс дорожки записью, в поле ключа которой во все разряды заносятся двоичные единицы и которая, таким образом, служит для опознания конца индекса дорожки. Поле данных этой записи игнорируется.

Номер дорожки	Запись № 1			Запись № 2		
	номер записи	поле ключа	поле данных	номер записи	поле ключа	поле данных
0	1	8	310018	2	8	310018
1	3	45	310101	4	45	310101
2	5	64	310201	6	64	310201
3	7			8		
4	9			10		
5	11			12		
6	13			14		
7	15			16		
—	17	11111111				

Рис. 7.14. Содержание индекса дорожки.

Пример 64. Пусть для файла выделен 31-й цилиндр и при первом его заполнении занятыми оказались дорожки 0, 1 и 2. Дорожки 8 и 9 оставлены для вытесняемых записей. Пренебрегая изображением данных, при заполнении дорожки указать только значения ключа записи.

Решение. Соответствующее заполнение файла показано на рис. 7.13.

Индекс дорожки для такого размещения записей показан на рис. 7.14.

Пусть теперь в файл поступает запись с ключом 10. Ее ключ сравнивается с ключами четных записей и находится ближайший больший ключ — 45, принадлежащий дорожке 1. Запись помещается в поле 1 дорожки 1, остальные записи подвигаются, а запись с ключом 45 вытесняется по адресу, даваемому записью R_0, в данном случае 310801 — адресу поля 1 дорожки 8 31-го цилиндра. Запись R_0 изменяет значение этого поля на 310802, а в записи 3 индекса дорожки ключ 45 заменяется ключом 40; в записи 4 адрес 310101 заменяется тем, который был в R_0: 310801; вытесненную запись надо искать именно по этому адресу.

Пусть теперь на этой же дорожке необходимо разместить запись с ключом 41. На дорожке ей места нет, так как ее ключ больше ключа крайней правой записи, но меньше 45 и, следовательно, на дорожку 3 ее помещать тоже нельзя. Поэтому она сразу должна пойти на дорожку переполнения, во второе ее поле. Записи 3 и 4 индекса дорожки примут вид

40 310101 45 310802

а запись R_0 будет содержать адрес 310803 и количество оставшихся байтов дорожки или дорожек переполнения.

Запись с ключом 41, попадая на дорожку переполнения, приобретает адрес 310801, т. е. адрес предыдущей вытесненной записи.

Поиск записей в индексно-последовательном файле основан на этом же принципе: по индексу цилиндра находят цилиндр, на котором следует искать запись, далее по индексу дорожки — дорожку, а потом ее ключ сравнивается с ключом первой из записей индекса дорожки, относящихся к этой дорожке. Это сравнение показывает, где искать запись: непосредственно на дорожке или на дорожке переполнения. В первом случае делается просмотр ключей всех записей на дорожке, а во втором — поиск идет по адресу, указываемому полем данных второй дорожки нужной пары. Если по этому адресу искомой записи нет, то там имеется адрес записи, вытесненной ранее, по которому мы за некоторое число шагов дойдем либо до искомой записи, либо до записи с нулевым адресом. (Это означает, что искомой записи в файле нет.)

Чтобы принять вытесненные записи в случае заполнения дорожек переполнения, создается так называемая *независимая область переполнения*. Индекс цилиндра также представляет собой отдельную область индексно-последовательного файла. Если файл состоит из многих цилиндров, то для ускорения поиска создается еще одна область — *область главного индекса*.

Поиск в индексно-последовательном файле, где значительная часть записей находится на дорожках переполнения, занимает больше времени, чем на аналогич-

ном файле с пустыми дорожками переполнения. Поэтому время от времени производят так называемую *реорганизацию файла* — перепись его на новое место с упорядочением по тем же ключам. После реорганизации все записи получают свое место на дорожках данных, а дорожки переполнения освобождаются.

Для многотомных индексно-последовательных файлов следует выделять непрерывную область данных: при переходе с тома на том она должна заканчиваться на последней дорожке последнего цилиндра (для УВВ ЕС-5052 — ЕС-5056 на дорожке 9 цилиндра 199), а начинаться на следующем томе на дорожке 0 первого цилиндра. Участки, выделяемые области индекса и независимой области переполнения, должны, таким образом, размещаться либо перед, либо после участка данных. Оглавление тома во всех томах следует размещать перед участком данных, и только в последнем томе оно может размещаться после участка данных.

7.4. Метки и их классификация

Метки файлов являются, с одной стороны, средством поиска этих файлов, а с другой — средством их защиты от случайного или преднамеренного разрушения.

Для аналогичных целей применяются метки томов — кассет магнитных лент, пакетов дисков. Типы, количество и размещение меток зависят от организации файлов и их носителей (см. пп. 7.2.1, 7.2.2).

7.4.1. Метки на магнитных лентах. В ДОС/ЕС допустимы файлы без каких-либо меток. Но если метки имеются, то начинаться лента должна с *головной метки тома*. Эта метка вместе с другими сведениями содержит *регистрационный номер кассеты*, распознаваемый системой и называемый иначе ее *внутренним номером* в отличие от *внешнего номера*, наносимого на кассету и предназначенного для ее опознавания человеком.

Хвостовая или конечная метка тома присутствует только в тех томах, которые имеют файлы, продолжающиеся на других кассетах. Применяются в ДОС/ЕС и *личные* метки тома.

Возможные типы меток и их коды приводятся в табл. 7.2. В табл. 7.2 n — номер метки. В ДОС/ЕС меток каждого типа может быть до восьми ($1 \leqslant n \leqslant 8$), но обрабатываются только первые метки ($n=1$).

Последовательность меток отделяется от файла маркером — особой записью, которую система создает и считывает аппаратными средствами. Также отделяются маркерами хвостовые метки одного файла от головных меток последующего. Таким образом, при наличии меток два соседних файла разделены тремя маркерами. За хвостовыми метками последнего файла ставятся два маркера.

Таблица 7.2

Тип меток	Кодирование меток			
	системных		пользователя (личных)	
	тома	файла	тома	файла
Головные (начальные)	VOLn	HDRn	—	UHLn
Хвостовые (конечные)	EOVn	EOFn	UVLn	UTLn

При открытии и закрытии файлов программист может задать режим обработки меток. На языке АССЕМБЛЕРА это достигается использованием соответствующих операндов в макрокомандах ввода — вывода, а на языке ПЛ/1 — выбором соответствующего режима описателя ENVIRONMENT.

Все стандартные метки на магнитных лентах ЕС ЭВМ (как системные, так и метки пользователя) имеют длину 80 байтов. Нестандартные метки пользователя могут иметь произвольную длину. На рис. 7.15, 7,16 показано содержимое первых меток из последовательности стандартных системных меток.

Обработка головной метки тома. Обращаясь к магнитной ленте, система прежде всего ищет запись VOL1, и если не находит, то прокручивает ленту до конца. Поэтому в материалах по математическому обеспечению ЕС ЭВМ существует независимая программа нанесения головной метки тома, работающая независимо от ДОС/ЕС. Найдя код головной метки, система считывает регистрационный номер тома, сравнивает его с соответствующим операндом оператора TLBL и в случае расхождения извещает об этом оператора.

Тип метки	Номера байтов				
	1—4	5—21	22—27	28—31	32—35
Головная метка тома	VOL1	Регистрационный номер тома	Не заполняется		
Хвостовая метка файла тома	HDR1	Идентификатор файла	Регистрационный номер файла	Порядковый номер тома	Порядковый номер файла
Головная метка файла	EOF1	▪	То же	То же	То же
Хвостовая метка файла	EOV1	▪	▪	▪	▪

Рис. 7.15. Метки на магнитных лентах.

Тип метки	Номера байтов							
	36—39	40—41	42—47	48—53	54—60		61—73	74—80
Головная метка тома	Не заполняется		Владелец тома	Не заполняется				
Хвостовая метка тома	Номер поколения	№ версии	Дата создания	Дата истечения срока хранения	Признак защиты	Счетчик блоков	Код системы (ДОС/ЕС)	Не заполняется
Головная метка файла	То же	То же	То же	То же	То же	Не заполняется	То же	То же
Хвостовая метка файла	»	»	»	»	»	Счетчик блоков	»	»

Рис. 7.15. Метки на магнитных лентах (продолжение).

Регистрационный номер тома должен совпадать с регистрационными номерами всех расположенных в нем файлов, кроме, может быть, первого, если его начало находится в другом томе. Остальную информацию, содержащуюся в метке тома, ДОС/ЕС игнорирует. Также игнорирует она и метки VOL2, VOL3, ..., VOL8, если они присутствуют.

Обработка головной метки файла. Головная метка файла помещается перед началом файла, а в многотомных файлах — и перед каждым его продолжением в новом томе. При наличии нескольких меток HDR1, HDR2, ..., HDR8 ДОС/ЕС обрабатывает только первую из них.

Разряды меток 5—21 система сверяет с операндом TLBL <'идентификатор файла'>, а при его отсутствии — с операндом <имя файла>. В случае надобности имя файла отыскивается среди операндов макрокоманды DTF или операторов DECLARE и DEFINE FILE.

Регистрационный номер файла (разряды меток 22—27) должны совпадать с регистрационным номером тома, за исключением случаев, когда в разрядах 28—31 указан номер тома, отличный от первого. При наличии в TLBL этих операндов для вводных файлов они должны согласовываться с указанными в метке. Аналогично сверяются номера файла, поколения и версии, и в случае обнаружения расхождений об этом извещается оператор. При отсутствии же в TLBL этих операндов их проверка не производится.

Для выводных файлов все перечисленные операнды записываются в метку, если режим создания файла предусматривает ее создание.

Поля метки <дата создания> и <дата истечения срока хранения> в случае выводных файлов заносятся в метку из системной области машины и операнда TLBL. Для файлов, подвергающихся обновлению, система допускает записи только в случае истечения срока хранения или после положительного ответа на запрос системы оператору. Этим система гарантирует защиту файла от непреднамеренного разрушения.

Из остальных полей обратим внимание на поле 55- — 60-го байтов, на котором в хвостовых метках записывается число блоков в файле. Наличие в этом поле пробелов гарантирует правильность счета числа блоков при обработке файла как в прямом, так и в обратном направлении.

Обработка хвостовой метки файла. Поля хвостовой метки, как правило, дублируют соответствующие поля головной метки и служат тем же целям, но при обработке файла в обратном направлении. Отличие в заполнении поля числа блоков объясняется убыванием содержимого счетчика при обработке файла в обратном направлении. Хвостовые метки выводных файлов создаются в ДОС/ЕС аналогично головным.

Обработка хвостовой метки тома. Хвостовая метка тома объединяет функции головной метки тома и хвостовой метки файла, не поместившегося в данном томе. Она содержит регистрационный номер своего тома, а не того, где начинается файл. Кроме того, она содержит счетчик блоков использующийся так же, как счетчик блоков в хвостовой метке файла.

Обработка меток пользователя. В ДОС/ЕС допускается использование программистом нестандартных меток. Для записи и чтения таких меток в программах пользователя на языке АССЕМБЛЕРА применяется операнд LABADDR макрокоманды DTF. При программировании на алгоритмических языках нестандартные метки не применяются. Аналогично используются в ДОС/ЕС и стандартные метки пользователя.

7.4.2. Метки на дисках. Метки на дисках служат тем же целям, что и метки на лентах. В ДОС/ЕС возможна обработка файлов на дисках и без меток, но только на уровне ФСУВВ. Для ЛСУВВ наличие меток обязательно. Как и на магнитных лентах, на пакетах дисков могут быть внешние метки — ярлыки с надписями, содержащими часть информации внутренних меток и в первую очередь регистрационный номер тома. Внешние метки рассчитаны на распознавание их человеком, внутренние — на распознавание системой. Метки на дисках также могут быть стандартными и нестандартными. ДОС/ЕС содержит программы обработки стандартных меток — создания их при выводе информации и проверки при вводе.

Стандарные метки делятся на метки *системы* и метки *пользователя.* На каждом пакете всегда имеется одна *метка тома.* Ее объем 84 байта: 4 байта — область ключа, а остальные — область данных. Как в области ключа, так и в первых четырех байтах области данных записывается код VOL1. Метка тома создается при *инициализации* диска специальной программой Эта метка

является третьей записью на пакете, а первые две содержат либо программу первоначальной загрузки, если пакет резидентный, либо остаются пустыми в противном случае.

ДОС/ЕС обрабатывает (считывает и записывает) только метки VOL1. Метки VOL2, VOL3, ..., VOL8, если они будут обнаружены на пакете, она пропускает. Метка тома на дисках (рис. 7.16) отличается от метки тома на ленте наличием поля ключа и *адреса оглавления тома* в байтах 12—21. Оглавление тома (VTOC) является первым из файлов, имеющихся на диске, и содержит метки всех этих файлов, в том числе и собственную метку.

Ключ	Данные						
	Номера байтов						
1—4	1—4	5—10	11	12—21	22—40	41—51	52—80
VOL1	VOL1	Регистрационный номер тома	Признак защиты	Адрес оглавления тома	Не заполняется	Владелец тома	Не заполняется

Рис. 7.16. Метка тома.

Стандартные метки файлов на дисках имеют длину 140 байтов, из которых 44 байта отводится под область ключа, а остальные под область данных. Каждая метка оформляется как запись. Первой из записей, имеющихся в VTOC, является метка оглавления тома VTOC. Она является частью определяемого ею файла. Все остальные метки от своих файлов отделены.

Стандартные метки имеют четыре различных формата соответственно выполняемым ими функциям. Основной является *метка формата 1,* создаваемая для каждого файла. Если файл состоит из четырех или большего количества участков, то для него создается *метка формата 3,* являющаяся продолжением метки формата 1. Если число участков в файле превышает 16, для него создается еще одна метка формата 3 и т. д. по необходимости. Метка формата 2 создается для индексно-последовательных файлов, а метка оглавления тома является *меткой формата 4.*

Основное содержание метки формата 1:

— идентификатор файла, в том числе номера поко-

ления и версии (байты 1—44),

— тип метки, т. е. номер файла (байт 45),

— регистрационный номер файла (байты 46—51),

— дата истечения срока хранения (байта 57—59),

— характеристики блоков и записей и сведения о наличии и месте ключа в них (байты 87—93),

— характеристики входящих в файл участков диска: тип, номер участка, нижняя и верхняя границы адресов участков (байты 106—115, 116—125 и 125—135),

— адрес метки формата 3 (если число участков превышает три (байты 136—140).

Поля, аналогичные последним четырем, составляют основную информацию метки формата 3.

Содержание метки формата 2 отражает специфику индексно-последовательных файлов: даются характеристики областей переполнения, индексных областей, отмечается наличие области главного индекса и пр. Все эти данные считываются из операторов EXTENT, описывающих файл, или из программ пользователя.

Обработка меток на дисках в ДОС/ЕС начинается с анализа метки тома. Регистрационный номер тома сверяется с соответствующим операндом оператора EXTENT для всех участков файла. Для файлов с последовательной организацией проверка идет в пределах одного тома, а для файлов с прямой организацией в ДОС/ЕС требуется доступ сразу ко всем томам.

При обработке метки файла формата 1 для каждого файла идет сравнение с его оператором DLBL, а для файлов, составленных из нескольких участков — с определяющими их операторами EXTENT. Как уже упоминалось выше, при отсутствии в операторе EXTENT операнда SYSxxx сведения о ЛУВВ система пытается найти в предыдущих аналогичных операторах, в макрокомандах DTF, операторах DECLARE, DEFINE FILE и т. д.

Метка файла находится с помощью поиска в VTOC, а адрес VTOC считывается из метки VOL1. Для поиска в VTOC используется операнд 'идентификатор файла' из DLBL, а если он отсутствует, то операнд <имя файла>. При сравнении размеров участков файла допускается, чтобы определенный в операторе EXTENT участок лежал внутри задаваемого в метке, но обратный случай, когда задаваемый в этом операторе участок выходит за пределы указанного в метке, считается ошибкой, о которой ДОС сообщает оператору.

Для обработки ДОС/ЕС меток пользователя для файлов на дисках ей необходимо дать указание об адресе обрабатывающей программы с помощью операнда LABADDR в макрокоманде DTF. Это средство может использовать только программист, работающий на языке АССЕМБЛЕРА.

Список рекомендуемой литературы

1. **Единая** система ЭВМ/ Под ред. А. М. Ларионова.— М.: Статистика, 1974.
2. **Операционная** система ДОС ЕС: Общие положения/ М. Р. Шура-Бура и др.— М.: Статистика, 1975.
3. **Языки** программироваиия ДОС ЕС. Краткий справочник/ **Авт.** Т. Д. Васючкова и др.— М.: Статистика, 1977.
4. **Лепин-Дмитрюков Г. А.** Программирование иа языке ПЛ/1 (для ДОС ЕС ЭВМ).— М.: Советское радио, 1978.
5. **Программирование** на языке Ассемблера ЕС ЭВМ/ З. С. Брич и др.— М.: Статистика, 1976.

Предметный указатель

БИБЛИОТЕКАРЬ

библиотек, корректировка 154, 157
— сжатие 171
— слияние 184
— создание и копирование 182
— удаление 162
библиотеки личные 12, 151
— модулей (абсолютных, объектных, исходных) 12, 152
— системные 12, 151
книга 153
модулей каталогизация 157
— обновление 154, 157
— переименование 164
— перфорация, печать 177
— удаление 162
модуль абсолютный 152
— исходный 153
— объектный 50, 152
— перемещаемый 50
модуля версия 5, 7, 168
— модификация 5, 7, 168
фаза 8, 15, 152
функции сервисные 175
функция распределения (памяти между разделами) 173
Инициатор одиночных программ 23, 24, 26
Операционная система дисковая ЕС ЭВМ (ДОС/ ЕС ЭВМ) 5, 7, 8
генерация 8, 9, 11, 13
загрузка первоначальная 10, 11, 13, 23
обработка пакетная 16
раздел памяти переднего плана 11, 14, 189
— — фоновый 11, 14
режим многопрограммный 10
— — с фиксированным числом задач 11
— однопрограммный 10
режимы ДОС/ ЕС временные 8
— — постоянные 9
— — стандартные 9
резиденция ДОС/ ЕС 12
устройства логические 8, 11, 12, 13, 14
— — программиста 11, 12
— — системные 11, 12
— периферийные 4, 7, 8
— физические 8, 12, 13, 14

РЕДАКТОР

адрес загрузки 187, 198
адреса смещение 146
модуль объектный вызывающий 193
— — нестандартный 191
— — смешанный 158, 193
— — стандартный 191
оглавление фаз 29, 189
операторы 195
программа оверлейная 150
режимы 8, 9, 32
секция программная 153
словарь внешних символов 193
— перемещений 192
таблица символических имен 144
— смещений 144, 146
точка входа 77, 187
фаза 8, 5, 152
— корневая 15, 19, 203
— самоперемещающаяся 15

СУПЕРВИЗОР

блок управления каналом 19
загрузчик системный 16, 21
канал мультиплексный 11
— селекторный 11
команды привилегированные 19
область связи 26
— сохранения 17
— транзитная 15, 16

планировщик каналов 16, 18, 210
прерывания 15, 16, 17
программа ВНИМАНИЕ 23, 24
— канала 19
слово адресное канала 18
— состояния канала 21
— — программы 17
транзиты 15
ядро 15, 16

УПРАВЛЕНИЕ ДАННЫМИ

блок 10, 212
дорожка переполнения 220
— разделения цилиндра 40
доступа способ последовательный 126, 130, 216
— — прямой 126, 216
записи блокированные 130
запись логическая 10, 213
— физическая 10, 213
— неопределенной длины 214
— переменной длины 214
— фиксированной длины 214
индекс дорожки 220
— цилиндра 220
маркер дорожки 213
— (марка) ленточный 213
метка тома 224
— файла 36, 66, 213, 229
набор данных 4, 10, 12, 13, 116
область данных 41
— индекса 41
— переполнения независимая 41, 223
оглавление тома 224
организации файлов способ индексно-последовательный 41, 123, 216, 220
— — — последовательный 123, 217
— — — прямой 123, 218
поле данных 221
— ключа 221
система управления вводом—выводом (СУВВ) 210
том многофайловый 10, 122
— однофайловый 10, 122
файл многотомный 10, 122
— однотомный 10
цилиндр меток 41, 151

УПРАВЛЕНИЕ ЗАДАНИЯМИ

аппарат назначений периферийных устройств 13
задание 10
задача 10
назначение режимов работы ДОС/ ЕС 32
назначение устройств 13, 14, 30
системная печать 12
системный ввод 12
— вывод 12
— перфоратор 12
управления заданиями директивы 5, 13
— — операторы 5
— — язык 5
шаг задания 10

ЯЗЫКИ ПРОГРАММИРОВАНИЯ

блок 51, 98, 103
— внешний 105
— начальный 51, 98, 103
— обычный 52, 103
— процедурный 50, 53, 105
блоков гнездование 104
выражение 62—65
— арифметическое 97
— булевское 64
— вторичное 64
— именующее 62
— первичное 65
— простое 63
— типа «массив» 85
выражение типа «структура» 85
— условное 62
Группа 94, 96
— итеративная 97
— простая 94, 98
данные типа «метка» 69
— — «указатель» 69
единица программная 104
заголовок процедуры 52, 105, 107
— — общего вида 107
— — функции 107
— цикла 102
значение (формальных параметров) 108
имя внешнее 192
— внутреннее 146
— входа 53

— входное 192
— массива 73
— переменной 66, 127
— — неиндексированной 66
— простое 55
— процедуры 52, 105
— — функции 52
— сложное 55
— структуры 75
— точки входа 77
— уточненное 85
— файла 33, 38, 116
констаита арифметическая 57
— действительная 56, 57
— логическая 57, 59, 60
— двойной длины 58
— обычной длины 57
— с плавающей точкой 57
— с фикснрованной точкой 57
— строчная 57, 59, 60
— — битовая 60
— — символьная 60
— целая 57
литерал 128
массив 49, 67
массива границы 73, 82
— размерность 74
— сегмент 82
метаязык 48
механизм активизации процедур 105, 112
множитель 65
— булевский 63
область общая в ФОРТРАНе) 74
обращение к процедуре рекурсивное 110
описатель 49, 80
— длины 71
— класса памяти 81
— области действия 125
— размерности массива 74
— размещения 83
— разрядности 70
— системы счисления 70
— строчных объектов 81
— файла 49, 122
описатель формы представления 70
— с помощью шаблона 71
описание (в ПЛ/1 — объявление) 67, 79
— массива 80, 73
— неявное 67, 81
— переключателя 77, 83
— переменной 69, 79
— процедуры 52, 106
— структуры 75, 83
— файла 120
— по шаблону 71
— явное 67
описаний (объявлений в ПЛ/1) область действия 72, 122
оператор 52
— безусловный 91
— ввода—вывода 5, 78, 89, 125
— возврата 52, 53
— вызова процедуры 78
— основной 78
— перехода 78, 86, 90
— присваивания 78, 84
— простой 78
— процедуры 78, 89, 104
— пустой 89, 78
— составной 51, 94, 98
— условный 78, 89, 91, 95
— цикла 78, 89, 97
параметр цикла 97, 102
параметры фактические 112
— формальиые 108
передача данных записями 122, 126
— — потоком 49, 122, 126
переключатель 77
переменная 66
— базированная 78
— действительная 69
— логическая 69
— металингвистическая 48
— обрабатываемая 72
— строчная 71
— — битовая 71, 133
— — символьная 71, 133
— типа метки 62
— — указателя 62
— управляющая 69
— целая 69
переопределение 78
переключатель 62
подпрограмма внешняя 106
— внутренняя 106
— общего вида 50, 106, 107
— функция 50, 106, 107
подструктура 75
порядок двоичный 59
— десятичный 59
процедура 50, 104
— без параметров 105
— внешняя 106

— внутренняя 106
— главная 51
— функция 52, 105
спецификация (формальных параметров в АЛГОЛе) 108, 109
список ввода 126
— вывода 135, 137
структура 49, 75
тело процедуры 52, 108, **110**
— цикла 89, 102
функции библиотечные **112**

Оглавление

Введение 3

1. Дисковая операционная система Единой Системы ЭВМ . 7
1.1. Единая Система ЭВМ 7
1.2. Структура и состав ДОС/ЕС 8
1.3. Физические и логические устройства 11
1.4. Аппарат назначений 13
1.5. СУПЕРВИЗОР 15
1.6. Прерывания и их виды 17
1.7. Планировщик каналов 18
1.8. Системный загрузчик 21
1.9. Связь оператора с ДОС/ЕС 22
1.10. Таблицы СУПЕРВИЗОРА 26

2. Управление заданиями 27
2.1. Основные функции УЗ 27
2.2. Подготовка программы к выполнению 28
2.3. Операторы назначения и переназначения УВВ . . 29
2.4. Установка режимов работы системы 32
2.5. Редактирование и запоминание информации о метках файлов 36
2.6. Составление комплексных заданий 43
2.7. Порядок следования операторов программы УЗ . . 45

3. Языки программирования ДОС/ЕС 46
3.1. Общие понятия 46
3.2. Выражения 62
3.3. Описания 67
3.4. Простые операторы 78
3.5. Условный оператор 91
3.6. Составной оператор (простая группа) 94
3.7. Оператор цикла (итеративная группа) 97
3.8. Обычный (начальный) блок 103
3.9. Процедуры (подпрограммы) 104
3.10. Библиотечные функции 112

4. Привязка языков программирования к ДОС/ЕС . . . 120
4.1. Описание файлов 120
4.2. Операторы ввода—вывода 125
4.3. Подготовка программ 142

5. БИБЛИОТЕКАРЬ 150
5.1. Структура библиотек и резидентного файла ДОС/ЕС 151
5.2. Состав БИБЛИОТЕКАРЯ и его функции 154
5.3. Формирование задания для БИБЛИОТЕКАРЯ . . 155
5.4. Корректировка 157
5.5. Сервисные функции БИБЛИОТЕКАРЯ 175
5.6. Создание и копирование 180

6. РЕДАКТОР **186**
6.1. Процесс редактирования 186
6.2. Структура объектных модулей 191
6.3. Разрешение внешних связей между модулями . . . 193
6.4. Операторы РЕДАКТОРА 195

7. Управление данными **210**
7.1. Система управления вводом — выводом 210
7.2. Организация данных на внешних носителях ЕС ЭВМ 212
7.3. Методы логической организации данных 216
7.4. Метки и их классификация 224

Список рекомендуемой литературы 232
Предметный указатель **233**

ИБ № 208

Лев Иванович ШАТРОВСКИЙ

МАКРОПРОГРАММИРОВАНИЕ
В ВЫЧИСЛИТЕЛЬНОЙ СРЕДЕ
ДОС/ЕС

Под редакцией проф. К. А. ПУПКОВА

Редактор М. С. Гордон
Художественный редактор А. Н. Алтунин
Обложка художника В. В. Кухта
Технический редактор Т. П. Сафонова
Корректор Л. С. Глаголева

Сдано в набор 16.08.78 Подписано в печать 27.12.78 Т-21878
Формат 84×108/32 Бумага тип. № 3 Гарнитура литерат.
Печать высокая. Объем 12,6 усл. п. л., 12,55 уч.-изд. л.
Тираж 50 000 экз. Зак. 856 Цена 90 коп.

Издательство «Советское радио», Москва, Главпочтамт, а/я 693

Московская типография № 10 «Союзполиграфпрома»
Государственного Комитета СССР
по делам издательств, полиграфии и книжной торговли.
Москва, М-114, Шлюзовая наб., 10.

Шатровский Л. И.

Ш 29 Макропрограммирование в вычислительной среде ДОС/ЕС. — М.: Сов. радио, 1979. — 240 с. с ил.

В пер.: 90 к.

В книге описываются средства дисковой операционной системы ЕС ЭВМ, позволяющие разрабатывать программные комплексы из отдельных модулей, создаваемых различными программистами. Излагаются методы использования компонентов ДОС ЕС: УПРАВЛЕНИЕ ЗАДАНИЯМИ, БИБЛИОТЕКАРЬ, РЕДАКТОР. Описаны языки программирования ДОС/ЕС: ФОРТРАН И ПЛ/1 путем сравнения их с АЛГОЛОМ-60.

Книга может служить пособием по программированию для ДОС/ЕС.

Ш $\frac{30502\text{-}012}{046(01)\text{-}79}$ 74-78 1502000000

ББК 32.81.
6.ФО.1

90 коп.



МАКРОПРОГРАММИРОВАНИЕ В ВЫЧИСЛИТЕЛЬНОЙ СРЕДЕ ДОС/ЕС